BOSTON UNIVERSITY

GRADUATE SCHOOL OF ARTS AND SCIENCES

Dissertation

AN ARCHITECTURAL APPROACH FOR MITIGATING NEXT-GENERATION DENIAL OF SERVICE ATTACKS

by

CODY DOUCETTE

BA in Computer Science, Boston University, 2014
MS in Computer Science, Boston University, 2014

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2021

Approved by

First Reader

John W. Byers, PhD
Professor, Computer Science
Associate Dean of the Faculty

Second Reader

Abraham Matta, PhD
Professor & Chair, Computer Science

Third Reader

Mark Crovella, PhD
Professor, Computer Science

To Erica

In loving memory of Allen C. Powell

ACKNOWLEDGMENTS

There are so many people that have supported me and helped me to get to this point. I would like to take a moment to thank each of them.

First, thank you to my wife and best friend, Erica. As if parenthood and a pandemic were not challenging enough, she was also subjected to the trials and tribulations of a husband finishing his dissertation. She rose to the occasion and supported me every step of the way, and for that I will always be grateful.

Thank you to my advisor, John Byers, for guiding me each step of the way. From the introductory class of his that I took ten years ago, to my time as an undergraduate research assistant, all the way to my status now as a PhD, I have always admired his knowledge, approach, and style in the academic arena. There is no one that I strive to impress more than John, and I hope this dissertation makes him proud of the work our research group has done.

I probably would not be here without the mentorship of Michel Machado. Michel took me under his wing during my undergraduate years and helped me grow to not just program better, but to think bigger. His visions for the potential of our research is inspiring, and I look forward to the years to come trying to accomplish those dreams alongside him. Thanks, Michel.

Thank you as well to my closest partner along this winding path, Qiaobin Fu. Qiaobin and I spent many, many hours building Gatekeeper from the ground up, and we could not have reached this point without each other.

Thank you to Dave Sullivan, who propelled my interest in computer science, taught me the value of sharing our knowledge with others, and gave me more opportunities than I can count throughout the "decade of Cody." Dave may have started out as my first CS instructor before becoming one of my most influential

mentors, but what I'm most thankful for is his friendship.

Thanks to Osama Haq and Fahad Dogar at Tufts University, who helped me learn the value of patience, persistence, and resilience in research endeavors. It took longer than we thought to publish our work, but the end product was superior for it, and I'm grateful for your partnership at every step along the way.

Thank you to my colleagues at BBN, who not only allowed me the freedom to finish my PhD while working full-time, but included me in engaging and challenging work and supported my new family and I throughout the past two years.

Thank you to all of the friends that have supported me, including Alexander Breen, Eli Saracino, Kylie Moses, Benjamin Brown, Andrew Tarrh, Sana Nagar, and a whole host of others that are too numerous to count, but which have made my ten years at BU an absolute pleasure.

Last but not least, thank you to my family, who have never held back in expressing their joy for me and my accomplishments. I hope this one makes you proud, too.

AN ARCHITECTURAL APPROACH FOR MITIGATING NEXT-GENERATION DENIAL OF SERVICE ATTACKS

CODY DOUCETTE

Boston University, Graduate School of Arts and Sciences, 2021

Major Professor: John W. Byers, PhD

ABSTRACT

It is well known that distributed denial of service attacks are a major threat to the Internet today. Surveys of network operators repeatedly show that the Internet's stakeholders are concerned, and the reasons for this are clear: the frequency, magnitude, and complexity of attacks are growing, and show no signs of slowing down. With the emergence of the Internet of Things, fifth-generation mobile networks, and IPv6, the Internet may soon be exposed to a new generation of sophisticated and powerful DDoS attacks.

But how did we get here? In one view, the potency of DDoS attacks is owed to a set of underlying architectural issues at the heart of the Internet. Guiding principles such as simplicity, openness, and autonomy have driven the Internet to be tremendously successful, but have the side effects of making it difficult to verify source addresses, classify unwanted packets, and forge cooperation between networks to stop traffic. These architectural issues make mitigating DDoS attacks a costly, uphill battle for victims, who have been left without an adequate defense.

Such a circumstance requires a solution that is aware of, and addresses, the architectural issues at play. Fueled by over 20 years worth of lessons learned from the industry and academic literature, Gatekeeper is a mitigation system that neu-

tralizes the issues that make DDoS attacks so powerful. It does so by enforcing a connection-oriented network layer and by leveraging a global distribution of upstream vantage points. Gatekeeper further distinguishes itself from previous solutions because it circumvents the necessity of mutual deployment between networks, allowing deployers to reap the full benefits alone and on day one.

Gatekeeper is an open-source, production-quality DDoS mitigation system. It is modular, scalable, and built using the latest advances in packet processing techniques. It implements the operational features required by today's network administrators, including support for bonded network devices, VLAN tagging, and control plane tools, and has been chosen for deployment by multiple networks.

However, an effective Gatekeeper deployment can only be achieved by writing and enforcing fine-grained and accurate network policies. While the basic function of such policies is to simply govern the sending ability of clients, Gatekeeper is capable of much more: multiple bandwidth limits, punishing flows for misbehavior, attack detection via machine learning, and the flexibility to support new protocols. Therefore, we provide a view into the richness and power of Gatekeeper policies in the form of a policy toolkit for network operators.

Finally, we must look to the future, and prepare for a potential next generation of powerful and costly DDoS attacks to grace our infrastructure. In particular, link flooding attacks such as Crossfire (Kang et al., 2013) use massive, distributed sets of bots with low-rate, legitimate-looking traffic to attack upstream links outside of the victim's control. A new generation of these attacks could soon be realized as IoT devices, 5G networks, and IPv6 simultaneously enter the network landscape. Gatekeeper is able to hinder the architectural advantages that fuel link flooding attacks, bounding their effectiveness.

CONTENTS

Acknowledgements **v**

Abstract **vii**

List of Tables **xiii**

List of Figures **xiv**

List of Symbols and Abbreviations **xvi**

1 Denial of Service in the Past, Present, and Future **1**

1.1 Overview 1

1.2 Mitigation Techniques 4

1.2.1 Threat Model and Goals 4

1.2.2 Commercial Solutions 7

1.2.3 Academic Solutions 8

1.3 Architectural Underpinnings 16

1.4 The Next Generation 19

1.5 Thesis Statement and Approach 21

1.6 Contributions 23

2 Gatekeeper **24**

2.1 Overview 24

2.1.1 Components 24

2.1.2 Step-By-Step Example 26

2.2 Design 28
2.2.1 Vantage Points 28
2.2.2 Gatekeeper Servers 34
2.2.3 Grantor Servers 39
2.2.4 Request Channel 41
2.2.5 Vulnerabilities 45
2.2.6 Architectural Properties and Deployability 46
2.3 Implementation 49
2.3.1 Packet Processing Framework 49
2.3.2 Functional Block Decomposition 51
2.3.3 Hardware Offloading 56
2.3.4 Software Techniques 58
2.3.5 Operational Features 59
2.4 Evaluation 61
2.4.1 Goals 61
2.4.2 Testbeds 61
2.4.3 Baseline Functionality 65
2.4.4 Effect of Policies 67
2.4.5 Performance Benchmarking 69
2.4.6 Cost Analysis 71

3 Policy Toolkit **75**
3.1 Overview 75
3.2 Policy Design 77
3.2.1 Decision Types 79
3.3 Writing Policies 81

3.4 Basic Policy Techniques 84
3.4.1 Host Lookups and Bogons 84
3.4.2 Port Lookups 87
3.4.3 Secondary and Negative Bandwidth 89
3.4.4 New Protocol Support: QUIC 93
3.5 Advanced Policy Techniques 96
3.5.1 Flow Capture and Analysis 96
3.5.2 Port Knocking 100
3.5.3 Load Balancing and Path Control 102

4 Defending Against Next-Generation Attacks 104
4.1 Overview 104
4.2 Crossfire Primer 106
4.2.1 Attack Summary 106
4.2.2 Architectural Advantages of Crossfire 109
4.2.3 Recorded Crossfire Attacks 111
4.2.4 Previous Attempts at a Solution 112
4.3 The Perfect Storm 114
4.4 Crossfire Defense with Gatekeeper 117
4.4.1 Measurement Study Setup 118
4.4.2 Link Map Disruption 121
4.4.3 Diversity of Cloud Paths From Gatekeeper 124
4.4.4 Moving Target Defense 132

5 Conclusions 135
5.1 Deployments 135

5.2 Closing Remarks 135

Bibliography **139**

Curriculum Vitae **150**

LIST OF TABLES

2.1 Gatekeeper priority assignment scheme 36

2.2 Instance types evaluated on AWS. 73

4.1 Summary of Crossfire construction 108

LIST OF FIGURES

1.1 Peak DDoS attack magnitude by year 1
1.2 Network capabilities design 10
1.3 Network filters design 14

2.1 Components of the Gatekeeper architecture 25
2.2 State transition diagram for flows in Gatekeeper 37
2.3 Request channel priority queue 44
2.4 Gatekeeper implementation block diagram 52
2.5 Grantor implementation block diagram 54
2.6 Amazon testbed topology 63
2.7 Legitimate file transfer time with low-rate policies 66
2.8 Legitimate file transfer time under SYN attack with secondary bandwidth 67
2.9 Legitimate file transfer time with negative bandwidth policies 68
2.10 Gatekeeper server throughput under maximum flow table churn . . 70
2.11 Legitimate file transfer time with varying Gatekeeper instance types and prices 74

3.1 Example geographic overview of flows associated to VPs 98

4.1 An overview of the Crossfire attack 107
4.2 Topological view of emulated bots and VPs 120
4.3 Handling of `traceroute` probes around Gatekeeper 122
4.4 Target links from measurement study 124

4.5 Degradation ratios by vantage point 126
4.6 Target links cut by flows from the various VPs (1 set of 20) 129
4.7 Target links cut by flows from the various VPs (3 sets of 10) 131
4.8 Heat map of persistent link overlap between VPs 133

LIST OF SYMBOLS AND ABBREVIATIONS

5G	Fifth generation mobile networks
AS	Autonomous System
AWS	Amazon Web Services
CDN	Content Distribution Network
(D)DoS	(Distributed) Denial of Service
DoC	Denial of Capability
(e)BPF	(Extended) Berkeley Packet Filter
EC2	Amazon Elastic Compute Cloud
ENA	Amazon Elastic Network Adapter
IDS	Intrusion Detection System
IoT	Internet of Things
IXP	Internet eXchange Point
LACP	Link Aggregation Control Protocol
LFA	Link flooding attack
PoP	Point of Presence
RSS	Receive-Side Scaling
VP	Vantage Point

CHAPTER 1

Denial of Service in the Past, Present, and Future

1.1 OVERVIEW

It is hard to overstate the impact that distributed denial of service (DDoS) attacks continue to have on the Internet. It is one of the top concerns of network operators (Akamai, 2019; Menscher, 2020), and by the numbers, it is easy to see why. Figure 1.1 shows the largest recorded attacks by volume for each year since 2002. Aside from a noticeable outlier in 2019, the trend is quite clear: peak attack magnitude is growing exponentially. Additionally, attacks are becoming more frequent and sophisticated, pushing mitigation systems to their breaking points and driving up the cost to defend against attacks.

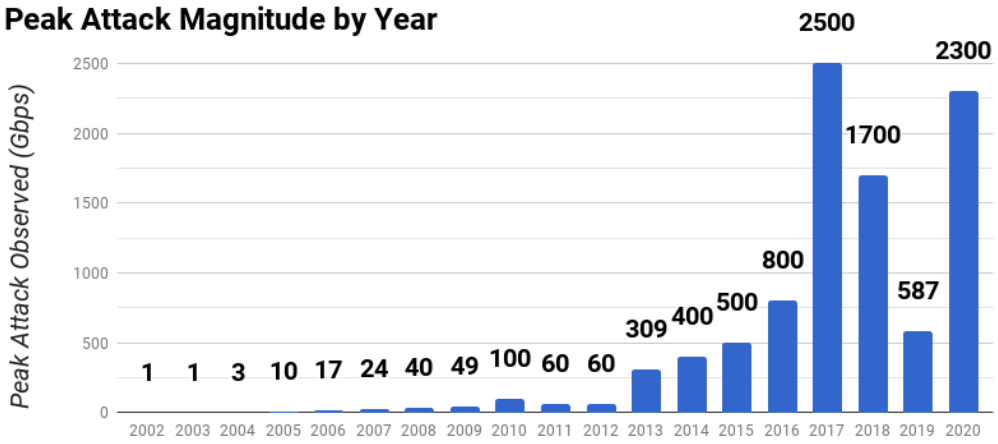


Figure 1.1: Approximate peak attack magnitude by year (Arbor Networks, 2014; NETSCOUT, 2019; Neustar, 2020; AWS, 2020b).

But perhaps an even clearer indication of the runaway nature of the problem is that the effects of DDoS attacks have entered the consciousness of the general public. The financial and social impacts of DDoS attacks are cited in high-profile

incidents, including the DDoS attack used to disrupt communications between protestors in Hong Kong (Marvin, 2019), as well as a 500% spike in DDoS attacks during the COVID-19 pandemic, as millions are forced to work remotely via the Internet (NexusGuard, 2020).

The economics of trying to defend against DDoS attacks is similarly grim. A report by Arbor Networks indicated that the cost incurred to victims from a major DDoS attack was over $220K (NETSCOUT, 2019). In contrast, attackers can easily launch a 125 Gbps DDoS attack for only several dollars (Makrushin, 2017). Additionally, the average cost of launching a DDoS attack will likely continue to fall through 2023, since the attack surfaces and resources leveraged by attackers are growing fast: the average broadband speed is more than doubling, and the number of IoT devices is growing nearly 2.4-fold (Cisco, 2020).

How have DDoS attacks become so damaging? Part of the issue is that there are certain architectural underpinnings of the Internet that enable DDoS attacks to be not just possible, but also potent. Both the academic networking community and the industry have implemented solutions to combat these problems, and although much progress has been made in identifying the salient issues and applying a patchwork of fixes, few have adequately addressed the holistic architectural advantage that attackers enjoy, and none have seen actual deployment. A solution is needed that both (1) addresses the architectural issues that fuel the ability of DDoS attacks to thrive in the Internet, and (2) that is able to be deployed with full benefits to drive adoption.

However, before we talk about a solution, in this chapter we will retrace our steps and think about DDoS attacks of the past, present, and future. To begin, we will define the threat model and goals of a DDoS mitigation system, and provide

a short tour of the relevant work in the commercial and academic arenas that have attempted to solve DDoS attacks. Next, we will explain the architectural properties of the Internet that have helped DDoS attackers to flourish. We then demonstrate the possibility for circumstances to worsen in the near future, and propose that Gatekeeper, a DDoS mitigation system, addresses the salient architectural issues, and can achieve the escape velocity needed to actually deploy elegant techniques from the literature in the Internet to combat the DDoS attacks of today and tomorrow.

1.2 MITIGATION TECHNIQUES

There have been many attempts at mitigating DDoS attacks, mostly split along two lines: commercial solutions and academic solutions. Both sides have made contributions to the state of the art in defensive systems, but there has been little cross-pollination between them, and there remains a large gap in the solution space. Elegant as they are, academic solutions have not been able to be deployed into the network due to the financial and technical barriers to making significant changes to the Internet architecture, which is often required for effective solutions. On the other side, the DDoS defense market is a lucrative industry, but the space of available products is quite limited and expensive, and the purveyors of these services are mostly only those who can use their distributed, Internet-scale infrastructure and services to absorb attacks (Nygren et al., 2010; Akamai, 2020; Cloudflare, 2020a; Imperva, 2020).

To dig deeper into the space of mitigation techniques, we will first define the threat model that mitigation systems are up against. Then, we highlight some solutions in both the commercial and academic arenas. Finally, we explain why there is still a need from the networking community for a cheap, comprehensive, and deployable defense for DDoS attacks.

1.2.1 Threat Model and Goals

Threat model. For this thesis, we define the threat model as follows:

- **Assumption 1.** *Attack locus.* We assume that attackers launch *infrastructure-layer* attacks, which are attacks that target the network and transport (L3 and L4) layers and infrastructure to deny access to legitimate clients by overwhelming the network bandwidth and processing capacity of victims. For

example: UDP fragment floods, DNS and NTP amplification attacks, TCP SYN floods, etc. Note that some infrastructure layer attacks, such as amplification attacks (Paxson, 2001; Czyz et al., 2014b; Rossow, 2014), use application layer vulnerabilities to trigger volumetric responses that ultimately overwhelm the infrastructure layer. As of 2017, infrastructure-layer attacks make up more than 99% of all DDoS traffic (Akamai, 2017). The remaining 1% consists of non-infrastructure, "low and slow" attacks, including:, but not limited to: (1) those that target applications or application data, such as HTTP `GET`, `PUSH`, and `POST` attacks; (2) Reduction of Quality (RoQ) attacks (Guirguis et al., 2004), which do not target the availability of victims, but rather orchestrate low-intensity requests that target system dynamics to destabilize the system and reduce service quality. Although some of the methods presented in this thesis may be applicable to these other non-infrastructure attacks, we do not explicitly address them.

- **Assumption 2.** *Attack type.* We assume that there is a set of infrastructure-layer attack types that is well-known to both the attacker and the mitigation system. Although this may give the mitigation system hints about the presence of an attack, the mitigation system does not know the exact parameters of these attacks, and cannot decide with absolute certainty whether a given flow is participating in such an attack. Additionally, we assume that there are novel infrastructure-layer attack types that the mitigation system does not know about *a priori*, but that this can be typified.

- **Assumption 3.** *Attack resources.* We assume that attackers have a fixed pool of resources (e.g., bots), and use those resources to launch attacks that are strong enough to achieve denial of service to legitimate clients, but no stronger. At-

tackers spread out their resources, such as per-bot attack rate, as much as possible to avoid detection.

- **Assumption 4.** *Attack source.* We assume that devices participating in an attack are out of the administrative control of the mitigation system, and that attackers (and their associated bots) hide their identities and try to place blame for attack traffic on unwitting, innocent clients by using source address spoofing.

A note on terminology: we will use the term *distributed* denial of service attack, since in the current Internet ecosystem of readily-available clouds and botnets, massively distributed attacks have become the norm. However, the principles outlined in this thesis are also applicable to non-distributed ("just" DoS) attacks.

Goals of mitigation. The basic goal of a DDoS defense system is to *mitigate* attacks, or sufficiently reduce the amount of attack traffic in the network to allow legitimate traffic access to services. To this end, the most effective DDoS mitigation system will:

- **Goal 1.** Minimize the amount of (wasted) bandwidth, memory, and CPU used in the hosts, routers, and other devices along the path from the attacker to the target.
- **Goal 2.** Minimize the time from attack onset to attack mitigation.
- **Goal 3.** Maximize the proportion of attack traffic that is mitigated.
- **Goal 4.** Minimize the amount of collateral damage, i.e., the amount of legitimate traffic that is reduced as a consequence of the system.

Note that these goals are not exhaustive, and it may not be possible to completely optimize (i.e., to absolute maximum or minimum) these goals. Because of Assumptions 2 and 4, which say that a mitigation system cannot perfectly decide whether a flow is participating in an attack and that the mitigation system has no authoritative control over the source, it is not feasible to mitigate 100% of attack traffic without any wasted resources or collateral damage.

Now that we have an understanding of the threat model and goals of mitigation, we will provide a brief survey of commercial and academic mitigation solutions.

1.2.2 Commercial Solutions

Commercially available DDoS mitigation solutions often provide redirection-based architectures as a paid service to their customers. For example, in the *clean-pipe* approach, a customer's traffic is redirected through a scrubbing center, which uses various analytic tools to identify malicious traffic and remove it from the data stream. The only traffic that reaches the customer's infrastructure is therefore "clean." Similarly, networks can contract the services of content distribution networks (CDNs) to gain access to their distributed and high-capacity infrastructure to dilute and absorb large attacks (Gilad et al., 2016).

Providers of DDoS mitigation services use these techniques and others, such as machine learning for traffic profiling, to perform attack detection and absorption. The exact services and options vary from provider to provider. Popular products in this space include Akamai Prolexic (Akamai, 2020), Cloudflare's DDoS protection tools (Cloudflare, 2020a), and Imperva Incapsula (Imperva, 2020).

Customers shopping for DDoS protection services have to consider multiple

issues when choosing a provider. First, they have to decide what forms of DDoS mitigation they are interested in, as some providers offer only application layer protections, while others offer transport, SSL, and infrastructure defenses as well. There can also be restrictions regarding how the customer can use the service; for example, infrastructure protection from Imperva Incapsula can only be applied for a single IP address or an entire C class (/24) network (Imperva, 2020). Additionally, customers who host latency-sensitive applications must be aware of the effects of the redirection through DDoS protection services on latency. Finally, most DDoS protection services use proprietary attack detection algorithms, giving the customers little control over how to analyze and prioritize traffic (Liu et al., 2016).

Cost is also a major factor. Although exact pricing is largely not publicly available, most enterprise-level tiers of protection are expensive[1]. Even the "free" tiers of DDoS protection that are often rolled into Web hosting or CDN services at no extra charge can have hidden fees, or have no service guarantees (Arazi, 2020). To help lower the financial burden, many providers offer *on demand* service alternatives (as opposed to *always on*), but this carries the risk of allowing an attack to succeed while the mitigation system is configured and the redirection mechanism is activated. Unfortunately, foregoing DDoS protection in today's Internet is not an option; according to an industry survey, each DDoS attack can cost the victim organization up to $50k (Corero, 2019).

1.2.3 Academic Solutions

Alongside commercial efforts, there has been a long trajectory of academic work to thwart DDoS attacks. While the threat of DDoS was known to the academic

[1]We provide more details about the cost of commercial DDoS protection services in Section 2.4.6.1.

community in the 1990s, the February 2000 attack by MafiaBoy that famously took down e-commerce sites such as Yahoo! and eBay (Moore et al., 2001) was a call-to-arms for network operators and researchers. Since that point, the community has taken a much broader interest in DDoS attacks and how to defend against them.

Denial of service is a multi-faceted problem. There is work in the realm of (1) attack detection, or how to differentiate malicious attacks from network misconfigurations, errors, and flash crowds, and to what extent such a differentiation matters; (2) attack mitigation, or how to stop attacks from disrupting service to legitimate users; and (3) attack accountability, or how to identify attackers and hold them responsible, potentially with performance penalties or even legal action.

This thesis sits squarely in the realm of attack mitigation. There have been many bodies of work in this field, including overlay-based architectures (Andersen, 2003) and blackholing approaches (Giotsas et al., 2017), but we will briefly highlight two of the largest swaths of academic work: network capabilities and filters.

1.2.3.1 Network Capabilities

IPv4 and IPv6, the ubiquitous network layer protocols in the data plane, are connectionless. Clients can transmit datagrams to servers without prior arrangement or permission. This design decision was made in an effort to keep the network layer communication open and simple, resulting in a best-effort service that has, in many ways, withstood the test of time.

From the perspective of defending against infrastructure-layer DDoS attacks, though, this is a vulnerability. Allowing network layer messages to reach the server necessarily means that network layer access has been granted, in the sense

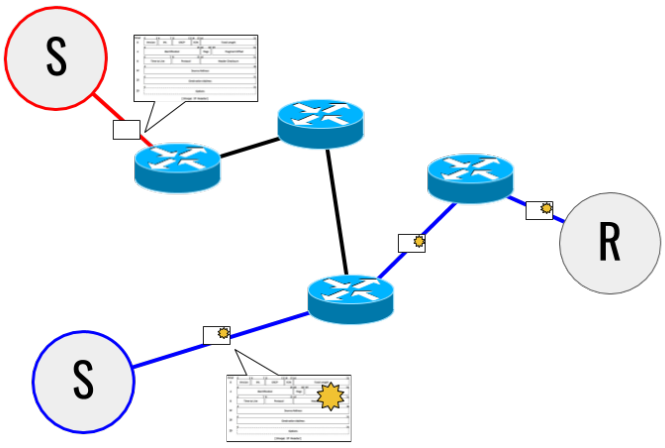


Figure 1.2: Transmission scenarios without (top) and with (bottom) network capabilities. Packets without capabilities are dropped at routers.

that the bits of the packet are being received, consuming some of the bandwidth of the server. Simple flooding attacks from small botnets can easily exhaust a server's network resources. This is the problem that *network capabilities* tries to solve (Anderson et al., 2004). In a network capability system, clients must explicitly receive an authorization token (a capability) to be able to send traffic. By default, some or all of the routers and hosts on the path from a source to a destination drop packets without a valid capability (Figure 1.2).

Essentially, network capabilities represent a connection-oriented network layer. But how could such an architecture work? Anderson et al. (2004) proposed that before being able to transmit normal traffic, senders must first request capability tokens from Request-To-Send servers. Once they obtain the token, they have explicit permission to send privileged traffic. Routers along the client-server path would act as capability verification points for such privileged traffic. Although this work provided the foundation for network capabilities, it was only a strawman design and was, to our knowledge, never implemented.

Other work then further developed capabilities as a new architectural technique. The Stateless Internet Flow Filter (SIFF) (Yaar et al., 2004) refined the network capability design, including removing the requirement for per-flow state at routers to validate capability tokens. The Traffic Validation Architecture (Yang et al., 2005) further enriched the state of the art by providing the first implementation of a capability system and evaluating it against various types of attacks.

At this point, there still existed practical and theoretical issues with network capabilities. On the practical side, there was still no workable solution for deploying a network capability system, including the fact that capability verification required upgrading routers along client-server paths. It seemed that the most realistic scenario was for deploying networks to only deploy the verification at their border routers, which still allowed attacks to consume resources up to the destination network. On the theoretical side, there was no feasible solution for protecting the capability setup (or capability request) channel. The request channel is a crucial part of the network capability architecture, since it represents the gateway for senders to able to obtain permission to transmit data, but is vulnerable to floods of requests from attackers that drown out legitimate requests. In fact, this flaw in the design of network capabilities became such a weakness that it threatened to derail the entire architecture.

Argyraki and Cheriton (Argyraki & Cheriton, 2005a) pointed out that this vulnerability, which they named the denial of capabilities (DoC) problem, presented a fundamental contradiction: if a mechanism existed to protect the capability setup channel, then the same mechanism could be used to protect the privileged traffic channel, therefore removing the need for capabilities in the first place. It was not until two years later that the Portcullis work showed that in fact, capability setup

traffic is different in nature than privileged traffic. Since only a single request packet is needed to obtain a capability token, the mitigation system just needs to provide a predictable and non-negligible probability that the sender's request packet reaches the receiver (Parno et al., 2007). By using computational puzzles to force fair sharing of the request channel, Portcullis had solved the DoC problem.

Additionally, there are more recent examples of work in the space of network capabilities. NetFence (Liu et al., 2010), allows receivers to use congestion policing feedback as capability tokens to suppress unwanted traffic, and also provides a time-based mechanism for guaranteeing a predictable capability setup request time, as opposed to the computational puzzles proposed by Portcullis. MiddlePolice (Liu et al., 2016) marries the concepts of cloud-based DDoS protection services with the destination-based control of network capability systems.

All in all, network capability systems would help mitigate DDoS attacks by largely replacing the free ability of attackers to transmit data to a server with a permission-based, connection-oriented network layer. However, none of the capability systems from the literature have been deployed, mostly due to the fact that there are few benefits to only a single network deploying these solutions (i.e., they all offer only incremental deployment benefits).

1.2.3.2 Filters

The filtering approach to DDoS mitigation is more in-line with the general principles of a connectionless network layer. By allowing traffic to be freely transmitted by default and only taking mitigating actions when issues are detected, filters represent an optimistic approach to DDoS defense, and are the *dual* to network capability systems.

The basic mechanism at play is a firewall: a set of filtering rules placed at one or more routers along the path from the clients to the target server (Figure 1.3). Each rule is typically specified in a declarative language that matches one or more features of a packet, and either software or hardware can apply the rules to received packets. Filters can drop or rate limit packets as the rule specifies. Challenges of filter systems include the fact that there is no single authority to trust when making filtering decisions upstream, the lack of source address verification to verify where filters should be placed, and state management issues for filters installed throughout the network.

In practice, firewalls and filters have been a part of production network software since at least the mid-1990s. For example, the Linux administrative utility for packet filtering, `iptables`, is capable of matching against any or all parts of an IP packet's five-tuple (protocol, source IP, destination IP, source port, destination port), along with other header fields as well as stateful filtering. `iptables` can be used to mitigate many network security events including some DDoS attacks, but it is insufficient to combat the scale and sophistication of attacks today, since filters are most effective when they are installed as close to the source of the attack as possible. Consider the bandwidth used by attack traffic at each hop between a malicious client and the target. The bandwidth used at each hop is bandwidth wasted, and the cycles used by routers on this path to process attack traffic is also wasted.

This is where the academic approach to network filters saw traction during the mid-to-late 2000s. How can filters be used as far upstream as possible to catch and stop as much traffic as possible? Pushback (Mahajan et al., 2002) was among the earliest efforts to capture the main challenges of filters, with a strawman design to

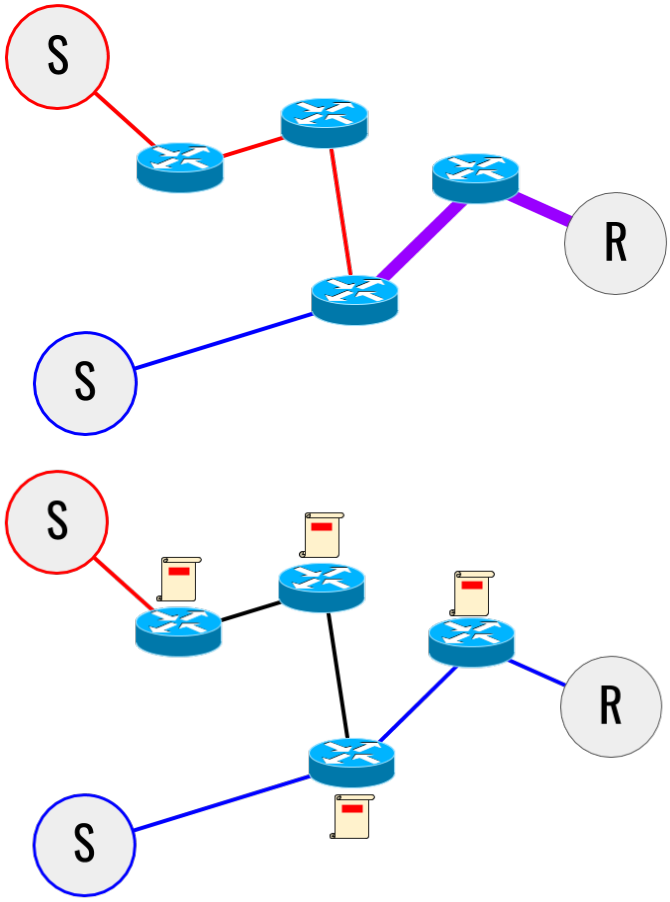


Figure 1.3: Transmission scenarios before (top) and after (bottom) filters are applied.

fingerprint traffic and apply filters. Filters can be applied both at the local router and up the path to the source by *pushing back* rules closer to the attack source. Argyraki & Cheriton (2005b) provided the Active Internet Traffic Filtering framework, which limits the amount of filtering state used in upstream routers by identifying the router closest to the attacking source that is willing to cooperate with filtering requests. Neither of these first two filtering proposals provided full implementations, and therefore were not deployed. Liu et al. (2008) contributed StopIt, a hardened filtering architecture that improved on previous work by adding mechanisms to resist (1) filter exhaustion attacks and (2) bandwidth flooding attacks that prevent the installation of filters. It also provided the first implementation (as opposed to simulation) of a filtering system, and offered a systematic comparison between filters and network capabilities. However, due to its only incremental deployment benefits, as well as its requirement for border routers to be upgraded to perform source authentication, StopIt was never deployed.

There are also recent proposals to collaborate with source networks to perform mitigating maneuvers with mechanisms other than filters, such as using rerouting techniques (Lee et al., 2013). However, since collaborative techniques require source networks to be incentivized to participate, which so far has not been shown to be feasible in the current Internet, there are high deployment hurdles for filtering architectures and none have been deployed in the wild.

1.3 ARCHITECTURAL UNDERPINNINGS

We now discuss the architectural underpinnings that not only enable infrastructure layer DDoS attacks to take place, but have also allowed them to flourish and evade mitigation for decades. First, a note on what we mean by the Internet architecture.

Definition. The *Internet architecture* is all of the complementary network layer protocols, infrastructure, and principles that make network interconnection possible:

- *Protocols*: data plane protocols such as IPv4 and IPv6 and their associated addressing schemes; control plane protocols such as IS-IS, BGP, RIP, etc.
- *Infrastructure*: the interconnected structure of hosts and routers which exist in access networks, transit networks, content delivery networks, points of presence (PoPs), peering links and Internet exchange points (IXPs), etc.
- *Principles*: the open Internet, a decentralized network of networks, the best-effort network-layer delivery service, the end-to-end principle, etc.

DDoS is an inherently architectural problem. To show this, we will describe four architectural issues that enable DDoS attacks, and explain why DDoS mitigation systems should address these issues to be most effective, referencing the goals defined in Section 1.2.1. While it may not be necessary to address all of the issues, each issue that is left unresolved limits the effectiveness of a mitigation system.

Architectural Issue 1. *Victim networks have little or no recourse with source networks.* Due to the decentralized design of the Internet – a network of autonomous systems (AS) with no central authority – there is no built-in mechanism for forging cooperation between networks to request attack mitigation from the source network.

The effect of Issue 1 is that it bounds the ability of a DDoS mitigation system to minimize the amount of wasted resources on the attacker-target path (Goal 1). To prevent the loss of network, memory, and CPU resources along the path, attacks should be mitigated as close to the source as possible. At present, the best way of doing this is with inter-AS cooperation between the victim and source networks.

Architectural Issue 2. *There is no mechanism to stop unwanted traffic.* Due to the open and connectionless properties of the network layer, traffic is by default permitted to transit the Internet and reach the destination network. Note that this issue is about the mechanism to stop traffic, which is distinct from Issue 1, which is about having the authority to stop traffic where it is most useful.

The effect of Issue 2 is that it guarantees that at least some attack traffic can reach a targeted server before mitigating maneuvers can take place, so any attack can be at least partially successful. This bounds the ability of a DDoS mitigation system to minimize the time to attack mitigation (Goal 2), and similar to Issue 1, limits its ability to minimize the amount of wasted resources (Goal 1).

Architectural Issue 3. *Sender identity cannot be verified.* Due to the lack of source address verification in the architecture, even if unwanted traffic could be stopped and mitigating maneuvers could be installed close to the source (Issues 1 and 2), there is no mechanism at the network layer to guarantee the identity of the sender of the traffic. Therefore, there is no guarantee that a mitigating maneuver itself would not be denying service to an unwitting legitimate client.

The effect of Issue 3 is that it interferes with the mitigation system's ability to accurately differentiate between attackers and legitimate clients, bounding its ability to maximize the proportion of attack traffic that is mitigated (Goal 3) and to minimize the amount of collateral damage in terms of legitimate clients (Goal 4).

Architectural Issue 4. *A victim network cannot compete with the capacity of Internet-scale attacks.* The Internet is a massive network of networks with billions of connected devices. Therefore, the capacity of a DDoS mitigation system in a singular victim network will almost always be exponentially smaller than the capacity of an Internet-scale botnet. Indeed, we know from analysis of the Mirai attack that it is possible to press devices into botnet service at the scale of the Internet (Antonakakis et al., 2017). The aggregate bandwidth consumed by a massive number of low-intensity, legitimate-looking attacking flows at the scale of even a fraction of the Internet will very likely be greater than the capacity of a defensive system.

The effect of Issue 4 is that it inhibits the ability of DDoS mitigation systems to reach all four defined goals. If the DDoS mitigation system itself is overwhelmed, perhaps by either bandwidth exhaustion or state exhaustion, there is not much that a system can do.

A solution to DDoS should be aware of these issues and address as many of them as possible to be most effective. Ideally, the solution should also be instantly and fully deployable, i.e., a deploying network should be able to reap the full benefits of the system alone and on day one. Otherwise, as has been seen with the great majority of DDoS mitigation proposals, systems that offer mere incremental deployment benefits will likely never see the light of day.

1.4 THE NEXT GENERATION

Taking the status quo of DDoS attacks and defensive systems into context, we now consider how the threat landscape may soon be shifting, yielding a new era of damaging attacks. Very large DDoS attacks can be quite scary for network operators, as during these events, operators must quickly perform triage to salvage services amid an avalanche of attack data, and with each second that passes and each flow that gets dropped, revenue is potentially lost.

Now, imagine the same event, but with a twist: no attack traffic reaches the victim servers and services. In fact, no traffic is reaching the victim at all. Operators have no visibility into what attack is happening or how to start to mitigate it. Imagine that such an attack could be achieved solely by using low-intensity traffic, so that even if an operator could peek at the traffic, there is no way to differentiate it from normal traffic. And imagine that the only way to stop such an attack would be to collaborate manually (e.g., over the phone) with other operators.

These are the properties of large-scale link attacks, specifically the Crossfire attack (Kang et al., 2013). Crossfire uses a massive and distributed botnet to orchestrate legitimate-looking flows to overwhelm a set of targeted links, upstream of the victim network and outside of the control of its operators. Crossfire has been described as “the most devastating and stealthy attack to date" (Gillani et al., 2015), and it is easy to see why: the operator does not know for sure that the attack is happening, there is no way to discern from where the attack is coming, and the operator cannot stop it without out-of-band help.

Such attacks are feasible in the Internet today, and similar attacks have already appeared in the wild (Bright, 2013). But now, there is a confluence of factors emerging in the Internet that could lead to a next generation of such large-scale link at-

tacks. Consider the following advances in technology:

- The proliferation of the *Internet of Things*, bringing about a new generation of "smart" Internet-connected devices.
- The rollout of fifth-generation (5G) cellular networks, giving mobile devices greater network capacity than ever before.
- The migration of networks to IPv6, revealing new, incrementally deployed infrastructure.

Client devices are becoming more plentiful and more powerful at the same time that many networks are undergoing infrastructural change. Could such an environment give rise to a new generation of stealthy and devastating attacks such as Crossfire? In this thesis, we consider a potential solution to DDoS attacks, including large scale link attacks: *Gatekeeper*.

1.5 THESIS STATEMENT AND APPROACH

The central thesis of this dissertation is that Gatekeeper is a deployable mitigation system that neutralizes the architectural issues that make DDoS attacks potent, in both the Internet of today and tomorrow. In order to formulate this finding, we answer the following questions as we proceed through the chapters:

1. What are the architectural issues at stake that make DDoS attacks possible and potent? In this chapter, we described the architectural underpinnings that have enabled DDoS attacks to grow to a top operational concern, while decades of work in DDoS defense have yielded few deployable results. Even worse is the fact that there exists a confluence of factors that might significantly amplify the damage of DDoS attacks in the near future. Such architectural issues need to be addressed in order for a DDoS mitigation system to be effective.

2. What would such a DDoS mitigation system look like? In Chapter 2, we describe the design, implementation, and evaluation of a DDoS defense system, Gatekeeper, which neutralizes the defined architectural issues. In addition, Gatekeeper is open source and deployable by a single network, meaning that unlike many DDoS solutions in the literature, it can see the light of day as a real-world deployment.

3. How is such a DDoS mitigation system managed? In Chapter 3, we explore the mechanism that governs the mitigation system: destination policies. Such policies control who is permitted access to the protected network's services, and how they do so, and ranges in scope from the ability to perform

simple lookups, to imposing punishments, supporting new protocols, and utilizing machine learning techniques.

4. What is the outlook for the future? In Chapter 4, we look to tomorrow, and theorize about whether Crossfire attacks are poised to take advantage of a confluence of factors to unleash devastating DDoS attacks in the near future, and what Gatekeeper can do in response.

To conclude, in Chapter 5, we discuss the ongoing deployments of Gatekeeper and future research directions.

1.6 CONTRIBUTIONS

Woven into the five chapters of this dissertation is a set of core contributions, including:

- An interpretation of the features of the Internet architecture that allow DDoS attacks to not just be possible, but to flourish
- The design of a fully deployable DDoS mitigation system, Gatekeeper, that leverages historical work in DDoS defense, and neutralizes the aforementioned architectural issues at play
- An open source implementation of Gatekeeper, built with performance and operational considerations in mind
- An evaluation of the effectiveness and efficiency of Gatekeeper, as well as an analysis of the cost of Gatekeeper compared to other mitigation systems
- A network operator toolkit for writing rich and expressive destination policies in Gatekeeper
- An analysis of the potential for a next generation of massive link attacks due to the confluence of IoT, 5G, and IPv6 entering the network landscape
- Open source implementations of the link map construction and target link selection algorithm described in Kang et al. (2013)
- A prescription for how Gatekeeper can respond to such next-generation attacks using cloud paths with a real-world evaluation using Looking Glass nodes.

CHAPTER 2

Gatekeeper

2.1 OVERVIEW

This chapter presents *Gatekeeper*, a DDoS mitigation system that (1) is fully deployable[1] and (2) addresses the network-architectural issues at play, incorporating many of the lessons learned by decades of DDoS mitigation research. Together, these properties enable Gatekeeper to bridge the gap between the research community and the real world DDoS protection market, and provide an avenue for a comprehensive DDoS mitigation system to be put into practice by a range of deployers, balancing cost with scalability.

2.1.1 Components

At its heart, Gatekeeper is a network capability system (Section 1.2.3.1). Previous proposals for capability-based systems failed to be deployed because in order to be successful, the deploying AS needed to control both ends of the path (in order to do capability management) or have trust arrangements with source networks. Gatekeeper circumvents these issues by leveraging geographically-distributed *vantage points* (VPs).

Between the VPs and the destination network, Gatekeeper provides a network-layer connection mechanism, which requires an authoritative entity in the destination network to explicitly grant access to client traffic before it can reach the intended destination server. This permission, or capability, takes the form of a special value in the IP header of packets sent along the path from the VP to the

[1] Such that the deploying entity can reap the full benefits of the system alone and on day one.

destination server.

Figure 2.1 depicts the general topology and highlights the two main components of Gatekeeper: *Gatekeeper servers* located in each contracted vantage point, and a *Grantor server* located in the data center of the defended AS. For simplicity, we assume that the AS deploys all its services in a single data center.

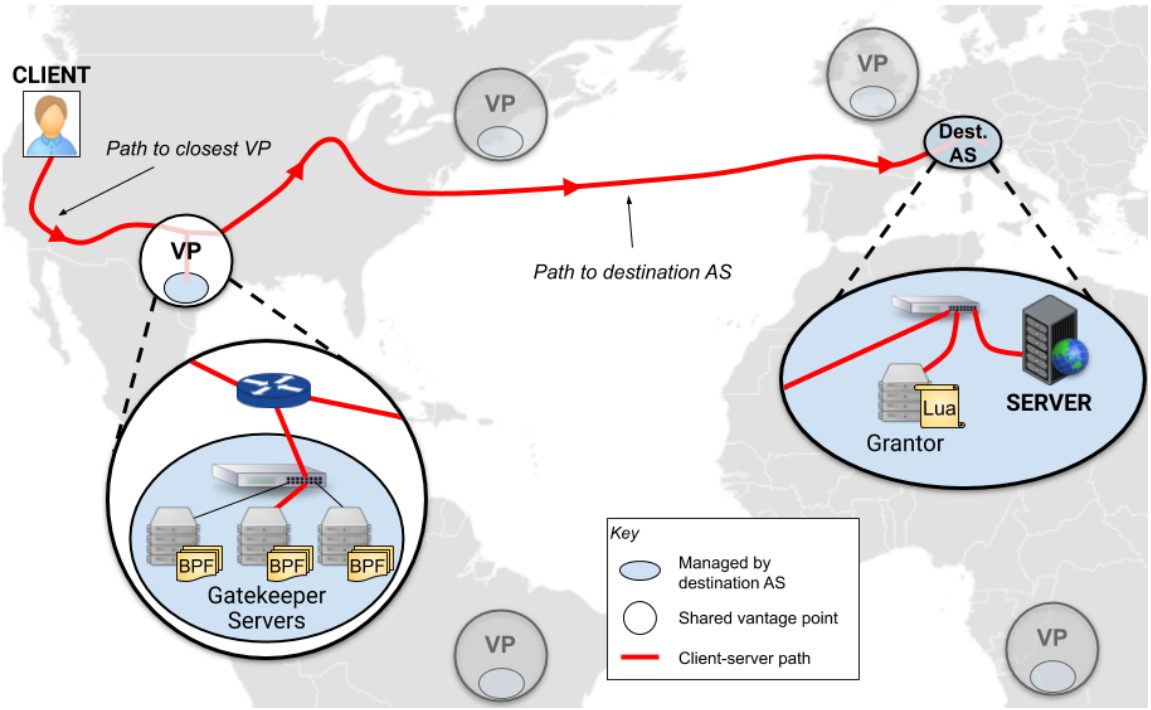


Figure 2.1: Components of the Gatekeeper architecture.

Each time a client tries to send traffic to a server in the destination network, the client's traffic gets forwarded to the closest VP. There, the VP performs network capability admission control. If a capability decision for the flow has not already been made, the first packet of a flow is marked as a request and is forwarded to the Grantor server, which will decide to either grant or decline the flow based on a defined policy. If the flow has already been previously granted, then rate-limiting state is updated, and if the flow has enough credits then the packet is sent along

the path to the destination AS, where it is decapsulated by Grantor and sent to the ultimate destination.

In other words, at a high level, Grantor provides the ability to perform centralized policy decisions, and Gatekeeper provides the upstream policy enforcement. Both ends of the this process – decision and enforcement – can be performed using *programs* instead of static, declarative rules.

2.1.2 Step-By-Step Example

To demonstrate how the components of Gatekeeper work together, we now describe an end-to-end example of a client initiating a TCP connection. Since Gatekeeper supports TCP/IP clients without modification, the client first sends a TCP SYN as normal. The SYN packet is forwarded to the closest VP, since all contracted VPs announce routes to the destination AS. Once received by the VP, the packet is transferred to the router of the destination AS, which in turn hands the packet to one of the Gatekeeper servers.

The Gatekeeper server checks whether there is state associated with the flow of the packet, and if there is, decides what to do based on that state. In Gatekeeper, flows are defined as the pair (source IP, destination IP).

In this example, we assume there is no state associated with the flow. In other words, a capability decision has not yet been made. The Gatekeeper server first allocates flow state that contains the arrival time of the SYN packet. Then, Gatekeeper encapsulates the packet using the IP-in-IP protocol, and fills the new IP header as follows: (a) the source IP address is the IP address of the server, (b) the destination IP address is the address of a Grantor server, and (c) the DSCP field is a value in the range $3-63$ as determined by a priority assignment algorithm. Higher

priority requests receive preferential service along the Gatekeeper to Grantor path during attacks.

The encapsulated SYN packet is then forwarded to the destination AS using a previously established tunnel. At the destination, the packet is delivered to the Grantor server, which then decides whether to accept the packet based on a policy defined by the deploying AS.

If the decision is to reject the connection, the Grantor server could do nothing, or send its decision back to the Gatekeeper server to avoid another query for a given period of time. If the decision is to accept, Grantor sends its decision back to the Gatekeeper server, which includes the maximum rate at which the sender can transmit (e.g., 1 Mbps) and a time limit for how long the decision is valid. Grantor then transmits the SYN packet to the target server.

After processing the SYN packet, the destination server replies with a SYN ACK that is sent directly to the source. The decision of the Grantor server arrives at the Gatekeeper server and the SYN ACK packet arrives at the source. The source will continue to send packets through the Gatekeeper server to the destination as the Gatekeeper server enforces the assigned rate.

2.2 DESIGN

We now present further details with regards to the four main components of Gatekeeper: vantage points, Gatekeeper servers, Grantor servers, and the request channel.

2.2.1 Vantage Points

Vantage points are a central piece of Gatekeeper's design. The term *vantage point* is commonly used in the network tomography community to refer to strategic points of the network from which to conduct measurements, or from which especially rich, diverse, or representative measurements can be made. However, in Gatekeeper, vantage points (VPs) are strategic locations in the network where a Gatekeeper server deployment is hosted. Common examples are Internet exchange points (IXPs), cloud data centers, and carrier hotels that host private interconnections. In this work, we will mostly use IXPs as exemplars for VPs, since the structure and characteristics of IXPs are well-documented due to their more public and transparent nature compared to clouds and private interconnects. Still, all such entities are considered vantages because they offer four benefits for DDoS mitigation systems, which are described next.

2.2.1.1 Benefits of VPs

First, VPs are often well-provisioned. For example, the switching fabric of clouds and IXPs are typically highly-redundant and composed of multiple aggregation levels across distributed colocation sites (Chatzis et al., 2013). This provides for high levels of resiliency and bisection bandwidth; some IXPs are already capable of processing traffic at peak speeds of more than 10 Tbps (IX.br, 2020). High peak

capacity is key to being able to mitigate tail DDoS attacks while minimizing collateral damage to clients.

The second advantage is that VPs are often available topologically close to source networks. According to various measurement studies (Gill et al., 2008; Dhamdhere & Dovrolis, 2010), the topology of the Internet is *flattening*, meaning that the Internet is shifting away from hierarchical routing through tiers of ISPs and transit providers and toward a mesh-like routing structure due to peering agreements between networks. Cloud providers participate too: Microsoft and Amazon peer at IXPs (AWS, 2020d; Yeganeh et al., 2019). Although transit networks still play a key role for reachability, the effect of this flattening is that VPs can act as insertion points for DDoS mitigation much closer to the source network than if hierarchical routing through transit providers was used. This is key to being able to fulfill the goal of minimizing the amount of wasted resources along an attacker's path to the target.

Third, VPs are globally distributed. By recent counts, there are over 800 IXPs distributed across over 130 countries in every continent except Antarctica (PeeringDB, 2020; Packet Clearing House, 2020). On the cloud side, Amazon's data centers are distributed across more than 20 geographically distinct regions, with 220 points of presence (PoPs) that span more than 40 countries across all continents except Antarctica AWS (2020c). The global distribution of VPs is key to Gatekeeper being able to handle Internet-scale attacks.

The fourth advantage is that leveraging existing infrastructure reduces the capital expenses for network operators, lowering the barrier to deploy services. Data centers in clouds and IXPs typically pay the rent and provide the power, cooling, and basic network connectivity for members. Participants just need to bring a

router to peer along with whatever network gear, compute, and storage is needed. Although members are charged fees for their participation, these much smaller operating expenses pale in comparison to the investment that would otherwise be needed to deploy vantage points in private facilities.

2.2.1.2 Placement of VPs

A key issue in designing a distributed Gatekeeper deployment is how to select the set of VPs across the Internet wherein Gatekeeper is deployed. Although the formulation and solution to such a deployment problem is outside the scope of this thesis, we will provide some preliminary thoughts about how to address it.

The VP selection problem is similar to the *cache deployment optimization problem* (Hasan et al., 2014), an offline economic and planning problem across relatively long (months or years) time scales, where CDN operators choose the set of networks in which to deploy CDN caches. This is in contrast to the older *cache location problem* (Krishnan et al., 2000), which focuses the cache placement issue from the perspective of a single network. The main constraints and trade-offs for such a problem are deployment cost for the CDN, balanced against the performance requirements of end-users. The VP selection problem is concerned with the following properties.

Multiple time scales. Similar to cache deployment optimization, the VP selection problem is long-term planning issue, since deploying network hardware to a VP requires investment and permanent maintenance. However, the elasticity of the cloud enables semi-permanent VP deployments that could be bootstrapped as a response to an attack or for other short time scales.

From-scratch vs. iterative. The VP selection problem is relevant for both first-

time and existing Gatekeeper deployers. Constructing an initial set of deploying VPs would be useful for first-time deployers, whereas an iterative version of the solution to calculate the marginal benefit of adding VP $n + 1$ would be beneficial for existing deployers.

Cost. An important input to the VP selection problem is the cost that a deployer would pay for a deployment in a given VP with a given level of resources. If cost were no objective, then the naive solution would be to simply deploy Gatekeeper at all candidate VPs. However, for each VP that is selected, the operator must pay for network gear (switches and servers), participation fees, and maintenance in the case of IXPs, and virtual compute, storage, and bandwidth charges in the case of clouds. Network operators are therefore interested in minimizing the management and operational costs that scale with the number of VPs, while maximizing the aggregate capacity of Gatekeeper, and also maximizing the spread of potential end-users (read: potential attackers) across VPs.

End-user and traffic spread. Typical metrics for cache placement problems are concerned with minimizing the total amount of traffic in the system and minimizing the delay experienced by end-users. For VP selection, end-user delay is a concern, but the main metric should be related to the potential peak traffic capacity, in hopes of spreading the potential amount of users and traffic to VPs proportionally based on their capacity. A traffic throughput history for each candidate VP would be a desirable input, but a proxy metric could be the number of IP addresses in the networks that would be anycast-served by the candidate VP. This could approximate the number of connected devices that would be served by the VP, but because of private address spaces and NAT, could also considerably undercount the actual figure. Ideally, the metric should also capture the potential attack traffic that is di-

rected through the VP, for which databases that describe network reputation could be used.

2.2.1.3 Requirements of VPs

In general, VPs have four requirements for basic operation: (1) computing capacity, (2) cheap ingress bandwidth, (3) BGP peering, and (4) private links to the protected network.

Computing capacity. A VP must be an insertion point for mitigation software to run. Nodes without general computing capacity, such as dedicated routers or other middleboxes, are insufficient to run Gatekeeper.

Cheap ingress bandwidth. A VP must support a pricing model where ingress bandwidth is either free or cheap. For example, implicit in the peering model of private interconnects and IXPs is that traffic is exchanged between participants for free. Similarly, many cloud providers use a pricing model where ingress traffic is free and users are charged for egress traffic only. This requirement is in place because of the nature of a DDoS mitigation system – in order to protect legitimate traffic at all times, the system is always on, so at a minimum legitimate traffic is always passing through the system. Charging for this traffic would be costly and impractical. At peak, during attacks, charging for ingress traffic would give attackers a direct lever on the price that victim networks pay.

BGP peering. VPs must support the ability to redirect traffic ultimately destined for the protected network through the vantage point first. In the current Internet, this is achieved through eBGP prefix announcements. Instead of announcing routes to a protected network's prefixes at traditional border routers, Gatekeeper servers announce the prefixes that they are configured to protect. In a

global deployment, this creates an anycast network, where client traffic is always forwarded to the nearest VP instead of directly to the protected network. Note that some network locations that would otherwise be good candidates for a VP, such as some cloud providers, do not allow customers to partipcate in BGP sessions, and therefore cannot be VPs.

Private links. VPs must be connected to the protected network by a private path that does not expose routable addresses to the open Internet. Otherwise, routers along the path would be susceptible to DDoS attacks themselves. To get the most protection along the path, physical links could be used (e.g., dark fiber), but that may be prohibitively expensive for some deployers. Instead, virtual links via tunneling (e.g., MPLS) using shared infrastructure would also be sufficient. However, using shared infrastructure raises the possibility of attacks on the links and routers on the path, even if the deployer's share of the resources is not directly reachable.

2.2.1.4 Relationship to Edge Computing

From a high level, Gatekeeper vantage points are in the domain of *edge computing*, a distributed computing paradigm in which the compute and storage power of the network edge is leveraged to mainly reduce application response time, but also to provide better scalability and reliability as well as offload computation from user devices when necessary (Satyanarayanan, 2017). There are two main ways of interpreting the "edge." One is as a separate, logical layer between access networks and the cloud, composed of one-hop-away devices such as fog computing nodes (e.g., routers and switches), mobile edge nodes (e.g., base stations), or cloudlets (dedicated "data centers in a box") (Bonomi et al., 2012; Satyanarayanan, 2017). The

second definition refers to distributed infrastructure further than one hop away, such as PoPs, IXPs, and private data centers where content caching, authentication, TCP termination, load balancing, DDoS mitigation, and other services can be inserted (Yap et al., 2017; Majkowski, 2017; Shirokov & Dasineni, 2018; Wragg, 2020). Gatekeeper fits squarely into this second definition, and is, to our knowledge, the first open source DDoS mitigation software for the edge.

Vantage points provide the infrastructure for Gatekeeper to effectively address the architectural barriers to combating DDoS. Next, we explore what role Gatekeeper servers play in that process.

2.2.2 Gatekeeper Servers

Gatekeeper servers are the main components deployed in VPs, and have the data plane responsibilities of (1) bookkeeping flow state and policy decisions, (2) enforcing policy decisions over flows, and (3) encapsulating traffic to be sent to the destination (protected) network.

2.2.2.1 Flow State and Processing

When traffic is redirected through the VP as a consequence of BGP announcements, ingress flows are load balanced between a set of one or more Gatekeeper servers located in the VP. Since Gatekeeper handles any IPv4 or IPv6 traffic, flows are defined as the (`Source IP`, `Destination IP`) pair, as opposed to common flow definitions that also include TCP or UDP port numbers. Gatekeeper maintains a flow table that maps incoming flows to flow entries, which can be one of four types:

1. *Request*. No policy decision for the flow has yet been made by Grantor, or a

previous policy decision for the flow has expired.

2. *Granted*. The flow is permitted to send traffic, and Gatekeeper will forward its traffic at a prescribed rate. Any packets from this flow beyond that rate will be dropped.

3. *Declined*. The flow is not permitted to send traffic. All packets from this flow will be dropped.

4. *BPF*. The flow is to be processed by a policy enforcement program installed at Gatekeeper. The program itself is (extended) *Berkeley Packet Filter* (eBPF) bytecode, which will forward or drop packets according to the specifics of the program.

If the flow does not have a corresponding entry in the table at all, a new entry of type *Request* is added to the table for the flow, and the packet is forwarded as a request to Grantor.

The original design of Gatekeeper only included the *Request*, *Granted*, and *Declined* flow states, which are sufficient for a network capabilities system. Later, the *BPF* flow state was added, which adds greater flexibility for operators to install arbitrary policy enforcement programs written in BPF to decide whether and how packets should be forwarded. For example, programs may use multiple bandwidth limits for different types of traffic within the same flow, which is not possible with the *Granted* state.

2.2.2.2 Flow Encapsulation and Channels

Request and granted packets are encapsulated so that they can be transmitted to a Grantor server in the destination network. There are two reasons for the encap-

Priority Value	Traffic Type
0	Legacy
1	Granted
2	Granted, renewing capability
3-63	Request

Table 2.1: Gatekeeper priority assignment scheme.

sulation. For request packets, a policy decision first needs to be made by Grantor before the original data packet is forwarded along to the destination server (or not, if the decision is *Declined*). Second, the encapsulation forces granted packets to be processed by Grantor, which allows a central point for measurement and enables the network operator to build policies around a global view of traffic. Optionally, the encapsulation of granted packets could be disabled, allowing for direct delivery of packets to their destinations, with the added benefit of reduced bandwidth consumption due to the lack of space taken up by encapsulation.

Although both request and granted packets are encapsulated and sent along the same path, they are processed in different logical *channels* according to a priority value assigned to each packet. Granted packets are high priority, and are given a vast majority of the path's available bandwidth, e.g., 95%. Request packets are given only a small fraction (e.g., 5%) of the available bandwidth and are scheduled using a special priority queue based on the packets' assigned priorities. Requests with higher priorities are serviced first. Further details about the request channel are in Section 2.2.4.

In theory, the priority assignment scheme could be designed in a number of ways, but in practice the inflexibility of the network layer forces us to work the priority scheme into the IPv4 and IPv6 protocols. Both of these protocols support a *Differentiated Services* (DS) field in the packet header that can accommodate a 6-

bit value. With six bits to work with, Gatekeeper uses priority assignment scheme shown in Table 2.1. Further details about how request priorities are assigned is described in Section 2.2.4. Central to Gatekeeper's design is the maintenance of flow states.

2.2.2.3 *Flow States*

When Gatekeeper receives a policy decision for a flow from Grantor, it updates the flow table entry. All policy decisions contain an expiration time as part of the response. Before a *Granted* or *BPF* flow's capability expires, Gatekeeper encapsulates a packet from the granted flow with a priority value reserved for granted flows whose capability is about to expire. This special priority value triggers a capability renewal from Grantor, avoiding delays or dropped packets that could happen as a result of a granted decision expiring and needing to be renewed from scratch. Otherwise, on expiry of a flow decision, flows go back to the *Request* state. A state transition diagram for flows in Gatekeeper is shown in Figure 2.2.

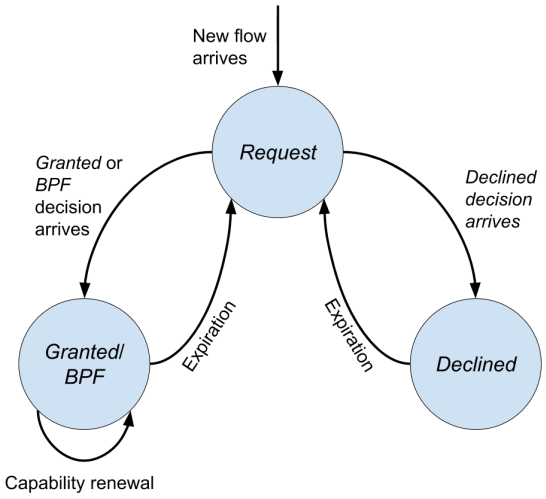


Figure 2.2: State transition diagram for flows in Gatekeeper.

To summarize, the flow processing algorithm is shown in Algorithm 1. For simplicity, the assignment of priorities to encapsulated packets is not shown.

Algorithm 1: Gatekeeper flow processing loop.

```
while True do
    pkt = rx_packet();
    flow = (pkt.srcIP, pkt.dstIP);
    flow_entry = flow_tbl_lookup(flow);
    if flow_entry != NULL then
        // Process flow according to flow state.
        if flow_entry.type == Request then
            tx_request_to_grantor(encapsulate(pkt));
        else if flow_entry.type == Granted then
            if flow_entry.avail_credits >= pkt.size then
                tx_granted_to_grantor(encapsulate(pkt));
            else
                drop_packet(pkt);
        else if flow_entry.type == Declined then
            drop_packet(pkt);
        else if flow_entry.type == BPF then
            send = flow_entry.enforcement_program(pkt);
            if send == True then
                tx_granted_to_grantor(encapsulate(pkt));
            else
                drop_packet(pkt);
    else
        // Process flow as a request.
        tx_request_to_grantor(encapsulate(pkt));
    end
end
```

2.2.2.4 Scalability

To maintain its effectiveness in the face of Internet-scale attacks, the design of Gatekeeper enables scaling along four axes:

1. Increasing the number of VPs. There are hundreds, if not thousands, of candidate locations for VPs across the globe. The more VPs a network utilizes, the more spread out ingress traffic will be, potentially diluting attacks.

2. Increasing the number of Gatekeeper servers in a VP. Multiple Gatekeeper instances can be deployed in a single VP with flows load balanced between them, enabling horizontal scaling.

3. Increasing the number of network ports used by each Gatekeeper server. Each Gatekeeper instance should be able to accommodate *bonded* network devices, allowing the network capacity of a single server to be scaled up without additional nodes.

4. Increasing the multithreading level of Gatekeeper servers. This allows Gatekeeper to scale-up the software processing power without additional hardware investment.

2.2.3 Grantor Servers

To complement Gatekeeper servers, Grantor servers are deployed in the protected AS to make the policy decisions that are enforced in the VPs. The responsibilities of Grantor servers include (1) decapsulating granted packets and sending them to their ultimate destination, and (2) running a policy decision program on request packets and informing Gatekeeper of such policy decisions. The details of the

policy program and decision-making are discussed in Chapter 3. While these responsibilities may seem like mundane technical details, they actually reflect three broader important roles that Grantor servers play.

2.2.3.1 Roles

Capability granting. Grantor is a trusted entity for capability granting and administers the capabilities for all protected hosts. Alternatively, capability granting could be performed by the individual destination servers themselves. However, using a separate authority to grant capabilities allows destination servers to remain unmodified, which is key for deployment and different from many previous network capability systems.

Centralized control. Grantor has centralized authority over the operation of the data plane, making it architecturally similar in spirit to Software-Defined Networking (SDN) (Casado et al., 2007; Yap et al., 2017), especially in the context of wide area networks (Yang et al., 2019). Using an SDN-like architecture simplifies network management for Gatekeeper, since operators only need to configure one (or a small set of) Grantor server(s) with the desired policy that is then enforced at the vantage points. In this view, Grantor servers are analogous to SDN *controllers* and Gatekeeper servers are analogous to SDN *switches*. Grantor servers install policy decisions on Gatekeeper servers using a protocol that is analogous to OpenFlow (McKeown et al., 2008). Additionally, the policy enforcement (BPF) programs that run at Gatekeeper servers are analogous to virtualized network functions (Han et al., 2015), as well as P4 data plane programs that can be run on SDN switches (Bosshart et al., 2014). The key difference between Gatekeeper and SDN is that Gatekeeper is scoped more narrowly to the issues and needs of DDoS miti-

gation, and therefore does not require the complexity and level of protocol support of OpenFlow and SDN switches and controllers.

Global traffic view. Since all request and granted traffic (not declined traffic) is sent through Grantor, it can act as a centralized location for measurement. This allows network operators to iteratively improve their network policies based on a global view of the traffic. If this service is not desired, Gatekeeper can optionally transmit all granted traffic directly to the destination servers, skipping the Grantor.

2.2.3.2 Scalability

Similar to Gatekeeper servers, Grantor servers provide scalability in several ways. Grantor servers can scale horizontally within a destination network, with flows load balanced between them. Grantor servers also have the same network and CPU scaling abilities as Gatekeeper servers, since they can bond network devices and utilize multiprocessing and multithreading.

2.2.4 Request Channel

The last major component of the Gatekeeper architecture is the request channel. The request channel is a key aspect of network capability systems, but is particularly prone to attacks, as described in Section 1.2.3.1. To summarize, care should be taken to ensure that the mechanism to request a capability token is not susceptible to DDoS attacks itself.

Gatekeeper takes three steps to protect the request channel: (1) allocating only a small fraction of the path's available bandwidth to the request channel, (2) assigning priorities to requests based on the time between successive packets in the same flow, and (3) dropping the lowest priority packets when the request channel

is overwhelmed. We now present details about each of these properties.

2.2.4.1 *Request Channel Bandwidth Allocation*

Separating low-priority request traffic from the high-priority privileged traffic has its origins in the earliest network capabilities proposals (Anderson et al., 2004). The reason for bifurcation is simple: the entity that performs connection management to protect destinations from unwanted traffic has no connection protection itself, so it is always possible for attackers to flood the connection setup mechanism. Gatekeeper combats this issue dividing the available capacity of the path from Gatekeeper to the destination network as approximately 95% granted traffic and 5% request traffic.

2.2.4.2 *Request Priority Assignment*

As previewed in Section 2.2.2.2, request packets are assigned a priority between 3-63, with higher priorities representing higher priority request traffic. The request priority assignment scheme stems from the results found in Portcullis (Parno et al., 2007). There, it is shown that a legitimate user (using Portcullis) can always successfully transmit a request packet in time bounded by the amount of attacker computation, since all clients must compute increasingly difficult proof-of-work puzzles to obtain increasingly high priority for their traffic.

However, since Gatekeeper is designed to be deployed by a single AS and Gatekeeper servers are therefore trusted entities, the priority assigned does not need to be cryptographically secure as it does in Portcullis. Therefore, Gatekeeper simplifies the proof-of-work mechanism to instead use waiting time as the metric to determine the priority assigned to packets.

The priority assigned to a request packet is $log_2(delta\_time)$. Therefore, flows that wait longer between successive packets are assigned higher priorities. This mechanism encourages exponential backoff, and is similar to the legitimate sender strategy outlined in Portcullis, in which legitimate senders will double the difficulty of the puzzles that they solve for every failed attempt. However, this is still not enough to mitigate the DoC problem: what if an attacker simply spams the request channel with low priority requests?

2.2.4.3 Priority Queuing Scheme

To prevent attackers from simply overwhelming the request channel with low-priority requests, nodes along the Gatekeeper-Grantor path (the Gatekeeper server and routers) must drop low-priority packets when their queues are full. To this end, Gatekeeper uses a priority queue that is implemented as a length-limited linked list of requests, sorted by priority (Figure 2.3). Additionally, Gatekeeper uses a constant-sized auxiliary array of references to the last (most recently added) request of each priority in the list, as well as references to the front of the list and back of the list. The priority queue supports the following operations, all of which run in time $O(1)$ relative to the size of the priority queue:

- *Dequeue highest priority packet*: the reference to the highest priority packet represents the next packet to be serviced by the packet scheduler, so the dequeuing operation simply removes the packet from the front of the queue. If there are multiple packets of the same such priority, the packet that has been in the queue the longest is scheduled.

- *Enqueue new packet, with available room*: if there is room available in the priority queue for another packet, then the packet's priority is looked up in the

priority reference table. If there are already packets of that priority in the queue, then the new packet is inserted at the end of the chain of those packets. If there are no packets of the priority in the queue, then the priority references are scanned to find the next lowest priority that is present in the queue. The new packet is then inserted after all packets of that next lowest priority.

- *Enqueue new packet, without available room*: if the queue is full, then the priority of the new packet is compared to the priority of the lowest packet in the queue. The packet with the lower priority between the two is dropped. If they have the same priority, the new packet is dropped.

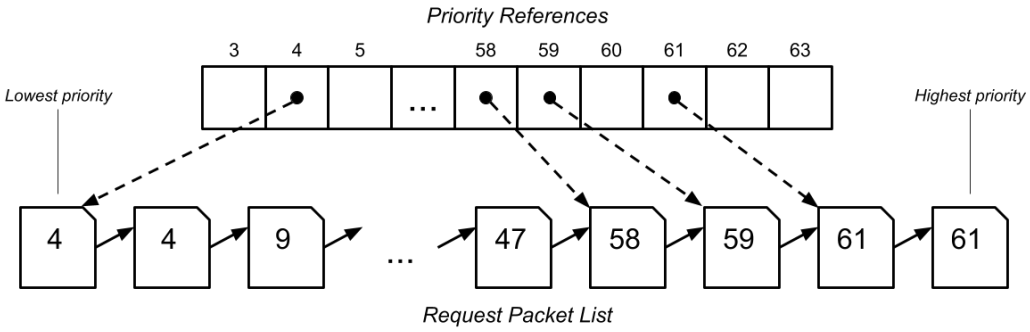


Figure 2.3: Request channel priority queue.

The three properties above – (1) limiting request channel bandwidth, (2) assigning request priorities based on waiting time, and (3) scheduling according to the priority queue scheme – are sufficient to guarantee the successful operation of the request channel according to the theorems proved by Portcullis. In particular, if a network policy cannot identify attackers, then a legitimate sender will wait, in the worst case, a time proportional to the number of attackers to have a policy decision installed at the corresponding Gatekeeper server, and that this result is optimal.

2.2.5 Vulnerabilities

In addition to the denial of capabilities problem, Gatekeeper must also be aware of and address potential vulnerabilities against itself.

Packet priority integrity. If a man-in-the-middle attacker were able to compromise a node on the path from the Gatekeeper server to the Grantor server, then that attacker could change the bits of the DS field of the outer header of Gatekeeper packets to change the priority from request to granted. This would allow the attacker's traffic to bypass some of the request channel throttling. However, this attack is limited in its effectiveness, since the request channel algorithm is also implemented at the Gatekeeper server, meaning that the 5% bandwidth limit and priority queueing scheme will at least be enforced at that point. Moreover, the effectiveness of the attack is bounded by the number of routers in the remaining part of the path after the priority is altered, since those are the only routers that would be processing the packet as granted. Finally, this attack does not permanently affect the state of the flow, which will remain in the request state, requiring the attacker to alter *every* packet's priority (and recompute every packet's checksum) in order to have an effect.

Spoofing policy decisions. A man-in-the-middle attacker could also compromise packets going in the other direction, by changing policy decisions sent from Grantor to Gatekeeper. Attackers are motivated to grant capabilities to their attacking flows, and to do so at high rates, to avoid the mitigation of attack traffic at Gatekeeper. This vulnerability can be solved by requiring digital signatures on policy decision packets, which guarantees the integrity of the decision, and by requiring signing entities (i.e., Grantor) to present a digital certificate, guaranteeing the identity of the signer.

State exhaustion. To calculate request packet priorities, Gatekeeper must keep track of the inter-request time for each flow. This means that Gatekeeper will keep per-flow state even during an attack, in which many packets with spoofed source IP addresses may be crafted to try to overwhelm the Gatekeeper flow table with many flow entries in the *Request* state. Gatekeeper handles this state exhaustion attack against the flow table with horizontal scaling and multithreading, where each thread in each Gatekeeper server is given a unique set of flows to be kept in its own flow table, which is not shared between threads or servers. Gatekeeper is also capable of using large flow tables, at least of size 2^{25} for each thread (Section 2.4.5). Also, old request entries are regularly pruned from the flow table, and even if there is not enough room in the flow table to hold a request entry, the request is still transmitted to Grantor to try to elicit a policy decision and at least allow the flow to make progress, even if under the bandwidth limit of the request channel.

Attacks against a VP. Attackers may target the links or other resources of the VP itself. VPs such as IXes and cloud data centers are generally well-provisioned due to the amount of traffic that they serve on a typical basis, but a motivated attacker with substantial resources could try to prevent packets from reaching Gatekeeper at all. However, the impact of these attacks is limited in Gatekeeper deployments that leverage multiple VPs, since routes would eventually be announced to direct traffic away from any affected VP. As a last resort, Gatekeeper can blackhole traffic at a VP to geographically bound the impact.

2.2.6 Architectural Properties and Deployability

Recall that Section 1.3 describes how an ideal mitigation system will address at least four core architectural issues to achieve a set of desirable goals for DDoS

defense, and do so in a deployable way. The design of Gatekeeper addresses all of these issues as follows:

Victim networks have little or no recourse with source networks. Communicating across trust domains at the level of trying to convince other ASes to take mitigating actions is an extremely difficult problem to solve. Gatekeeper circumvents this issue by using a geographic deployment that approximates the topological position of source networks with VPs. Therefore, Gatekeeper foregoes the requirement to collaborate with other networks to stop DDoS attacks while still achieving the goal of minimizing the amount of wasted resources on the attacker-target path.

There is no mechanism to stop unwanted traffic. By realizing a connection-oriented network layer with capabilities, Gatekeeper provides the missing shutoff mechanism to stop unwanted traffic. By being able to completely cut off any flow from reaching the network, rate limiting questionable flows immediately, and revoking capabilities, Gatekeeper minimizes the time to mitigate attacks.

Sender identity cannot be verified. Gatekeeper does not provide sender identity verification *per se*. However, it does provide extra topological information by virtue of the flow's entry into the protected network at a particular VP. This information can be used to detect flows with spoofed source addresses (Section 3.5.1.2). Additionally, Gatekeeper can enforce certain lightweight authorization mechanisms such as port knocking (Section 3.5.2). Policy techniques such as these allow Gatekeeper to differentiate between attacking flows and legitimate flows, maximizes the proportion of attack traffic that is mitigated and minimizes the amount of collateral damage to legitimate clients.

A victim network cannot compete with the capacity of Internet-scale attacks. Although Gatekeeper protects only a single network in current deployment mod-

els, it can achieve an Internet-scale deployment through its usage of distributed vantage points. It is still true that with enough money and bots, an attacker can overwhelm any DDoS mitigation system, Gatekeeper included. However, Gatekeeper's ability to scale along four axes (more: VPs, Gatekeeper servers, network devices, lcores) makes global deployments of Gatekeeper suitable for protection at the scale of multiple Tbps. This capacity at least matches the magnitude of the largest known attacks.

Deployability. Previous solutions were hampered by deployability issues, such as requiring extra-architectural mechanisms or non-incentivized collaboration between networks. However, Gatekeeper operates completely within the current Internet architecture by leveraging the existing infrastructure of globally distributed vantage points. Such vantage points also enable Gatekeeper to circumvent the necessity of mutual deployment with other networks, since it gives the deploying network a presence close to traffic sources to enforce network capabilities.

2.3 IMPLEMENTATION

Gatekeeper was designed to be deployable, and its implementation reflects that. It was built with performance, scalability, and operational requirements in mind. More than just a research prototype, the Gatekeeper software product combines advances in packet processing technology together with hardware offloading and a suite of desirable features for network operators. We now present an overview of the implementation details of Gatekeeper, and reflect on the implementation of the design decisions that drive Gatekeeper to be deployable.

2.3.1 Packet Processing Framework

A key concern for Gatekeeper is its dependence on, and its choice of, a packet processing framework. Typical operating system network stacks are well-suited for generality and are acceptable for a wide range of applications. However, they are too slow to keep pace with high-speed networks, and are not ideal for the packet and bandwidth processing needs of a DDoS mitigation system. Several packet processing frameworks such as netmap (Rizzo, 2012), Intel DPDK (DPDK, 2020), Fastclick (Barbette et al., 2015), `PF_RING` (PF_RING, 2020), and PacketShader I/O (Han et al., 2010) have been proposed to enable network applications to skip the overhead of the OS network stacks and provide line-rate network I/O. For Gatekeeper, Intel's Data Plane Development Kit, or DPDK, was the best choice. DPDK provides excellent performance in terms of throughput and packet processing latency (Gallenmüller et al., 2015), and its development is stable and driven by many industry collaborators.

At its core, DPDK is a set of libraries that accelerates packet processing workloads running on a wide variety of CPU architectures. DPDK enables link layer

frames to bypass the kernel networking stack and be delivered directly to applications. The main advantage of kernel bypass techniques like this is speed: traditional kernel networking stacks are comparatively bloated to provide generality of processing across a variety of link layer, network layer, and transport layer protocols. In DPDK, applications are essentially handed a simple buffer of bytes for each frame, and any protocol-specific processing must be implemented by the application developer. For example, there is no TCP implementation in DPDK, so DPDK cannot easily be used to manage connections as a TCP endpoint. However, a DPDK application could easily and efficiently sample TCP packets for measurement purposes, or perform transport layer load balancing. For a DDoS mitigation system like Gatekeeper, the speed advantage far outweighs the loss of generality disadvantage.

Gatekeeper heavily relies on three key features in DPDK: (1) NUMA-aware memory management, which can reduce the memory access latency by allowing CPU cores to access local memory instead of remote memory; (2) burst packet I/O, which allows Gatekeeper to receive and send packets in batches, reducing the per-packet cost of accessing and updating queues; (3) lockless rings, which provide an efficient concurrency control mechanism packet buffer allocation and inter-thread communication. More details about the software and hardware techniques used by Gatekeeper are given in Section 2.3.4 and Section 2.3.3, respectively.

An important piece of terminology in DPDK parlance is *lcore*, which DPDK defines as a logical execution unit of the processor, also known as a hardware thread. The Gatekeeper software is decomposed into specialized blocks, each of which is mapped to (at least one) lcore. We now describe each of these components and their role in the system.

2.3.2 Functional Block Decomposition

Gatekeeper and Grantor servers are implemented as a single piece of software, composed of *functional blocks* that represent the core components and services of the system. Configuration settings determine whether the executable runs as Gatekeeper or Grantor. There were three factors influencing the decision to decompose the systems into functional blocks in a single executable. First, having a single piece of software simplifies the engineering, configuration, and administration of the system. Second, it provides for separation of concerns and modularity, which allows the re-use of blocks that are common to both Gatekeeper and Grantor. Finally, it allows fine-tuning of system performance; each block is given one or more DPDK lcores, so operators can easily scale-up the blocks that implement data plane operations without wasting resources on low-bandwidth control plane operations.

Figures 2.4 and 2.5 present block-level views of the implementations of Gatekeeper and Grantor. Notice that the model of a Gatekeeper server has two physical NICs. One interface is connected to the VP router or switch, which we denote as the front interface of Gatekeeper, and one interface connected to the private link(s) to the AS deploying Gatekeeper, which we call back interface of Gatekeeper. Gatekeeper requires separate interfaces because otherwise, if there was only a single network interface, a Gatekeeper server under attack may not receive policy decisions coming from Grantor servers due to the saturation of the interface. In contrast, Grantor servers each have a single interface and connect to the network segment of the destination network's servers.

Each rounded-edge block in the diagram represents an lcore. The GK, GT, and SOL blocks are able to be mapped to multiple lcores, so they are shown as stacks of rounded-edge blocks. All other functional blocks are assigned at most one lcore.

Each arrow leaving or arriving at a network interface represents an RX (receive) or TX (transmit) queue on the interface, respectively. Notable hardware and software features are labeled, which include (1) *RSS*, receive-side scaling, which allows received packets to be spread among multiple queues on the same NIC for load balancing purposes; (2) lockless rings for message passing between blocks; (3) filters, or declarative traffic rules, which enable Gatekeeper to map certain protocols to functional blocks; (4) the KNI, or kernel-NIC interface, a way for DPDK to interface with the Linux kernel through a network device; and (5) a Unix socket, which provides the means for Gatekeeper to communicate with a client program for runtime configuration. More information about these components are presented alongside their associated functional blocks, which are described next.

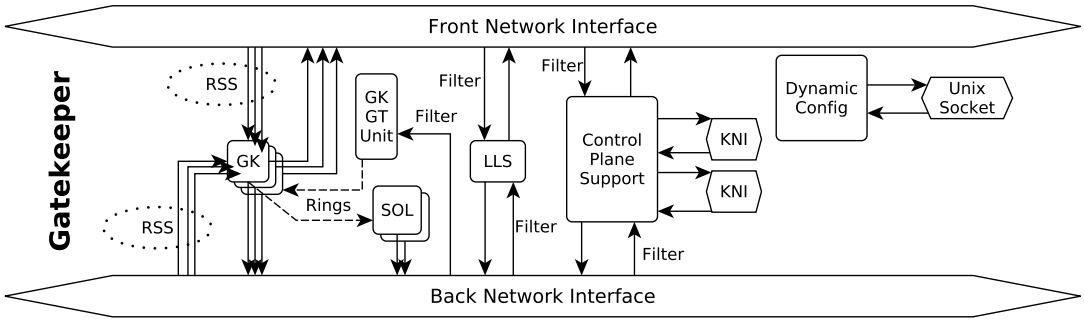


Figure 2.4: Gatekeeper block diagram.

2.3.2.1 GK block

The Gatekeeper (GK) block is the main component of Gatekeeper servers, as its task is to accept incoming packets, perform lookups that map flows to policy decisions, and queue requests and granted packets for transmission to Grantor. It is in the data plane and can scale across multiple GK *instances*, each with a dedicated lcore. Gatekeeper utilizes RSS to distribute incoming packets among the

GK instances. Each instance processes a queue whose packets have unique pairs of source and destination addresses. RSS provides the guarantee that packets belonging to the same flow will be directed to the same RX queue, and therefore the same GK instance, in the order that they arrive at Gatekeeper. Therefore, each GK instance maintains its own lockless flow hash table whose keys are these (source, destination) pairs. Because of its position on the front lines of the data plane, meaning it will bear the brunt of attacks, the GK block uses prefetching, batching, and other memory optimization techniques to maintain performance (Fu, 2020).

The main algorithm of the GK block is to read a batch of packets from the front interface and perform flow lookups over the batch. Packets whose flows have entries in the table are handled according to the flow entry – drop, forward, or apply a BPF program. Packets whose flows are not in the flow table are looked up in an LPM table that defines the forwarding action. Most flows are forwarded to Grantor (first as a request), but GK blocks also support Gatekeeper bypass for incremental deployment and peering traffic. For these flows, the GK block simply forwards the packets to the opposite interface.

Additionally, Gatekeeper implements the full-functionality option of maintaining the ECN bits for IP-in-IP packets (Floyd et al., 2001), which maintains any congestion management occurring between the source and destination.

2.3.2.2 GT block

The Grantor (GT) block is the main component of Grantor servers, as its task is to accept incoming packets from Gatekeeper servers, issue policy decisions for requests, and forward granted packets to their ultimate destination. It is in the data plane and can scale across multiple lcores. Chapter 3 provides a thorough

overview of the role of the policies that the GT block manages.

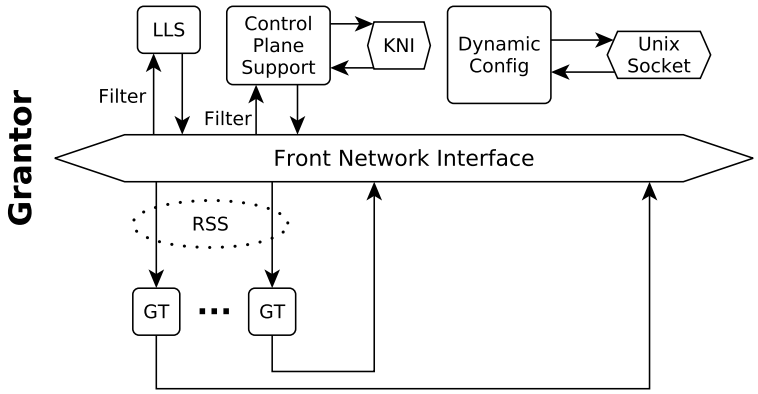


Figure 2.5: Grantor block diagram.

2.3.2.3 Solicitor (SOL) block

The SOL block is responsible for rate limiting and sending request packets. Requests are sorted by priority and only permitted a fraction of the link capacity between Gatekeeper and Grantor, and the SOL block enforces these limits. The priority queue (fully described in Section 2.2.4.3) is implemented as a length-limited linked list of request packets, indexed by an array whose elements are references to the portion of the linked list that holds packets of each priority, providing constant time insertion, dequeuing of the highest priority request, and deletion of the lowest priority request when the queue is full. The SOL block only runs when the Gatekeeper program is being run as a Gatekeeper server. Multiple SOL instances are used to spread the load from multiple GK instances.

2.3.2.4 GT-GK Unit (GGU) block

The GT-GK unit (GGU) processes all policy decisions that Grantor servers send to a Gatekeeper server. Each packet coming from a Grantor server carries a set of

decisions. The GGU block demultiplexes each decision to the GK instance that is responsible for the flow in question, according to its RSS hash. Note that GGU only operates on the "back" interface of Gatekeeper (the interface that leads to the protected AS).

The GGU loads a set of GT-GK packets from the back NIC. For each packet in this set, it loops over each policy decision on the packet, and for each decision does the following: (1) identify which GK block is responsible for the pair (source, destination) addresses in the decision; (2) obtain the mailbox of that GK block; (3) send the policy decision to the GK block via the mailbox.

2.3.2.5 Control plane support (CPS) block

Gatekeeper servers must be able to use routing daemons to peer in IXPs. Similar to Google's Espresso (Yap et al., 2017), we implemented this peering functionality in Gatekeeper servers. Instead of adding support for a multitude of common control plane protocols (e.g., BGP, OSPF, and IS-IS) in Gatekeeper directly, we enable network operators to use existing routing daemons and management tools by leveraging the DPDK kernel-NIC interface (KNI) library. In our deployment, the CPS block uses the popular BGP speaker BIRD (BIRD, 2020).

2.3.2.6 Link layer support (LLS) block

The LLS block has the responsibility of handling all link layer protocols and address resolution services, such as ARP, Neighbor Discovery, and Link Aggregation Control Protocol (LACP). Instead of resolving IP addresses to MAC addresses on demand, other functional blocks must register the IP addresses that they are interested in, and the LLS block keeps IP address-to-link layer address maps updated.

To guarantee that other blocks are not kept waiting for resolution requests, the components that query the LLS block will drop packets directly instead of waiting for a resolution if the map is not current.

2.3.2.7 Dynamic configuration block

Both Gatekeeper and Grantor have two types of configurations: static and dynamic configurations. When Gatekeeper boots, it configures the network and individual functional blocks using the static Lua configuration files. Several features, such as jumbo frames, MTU, IP/UDP header checksum offloading and VLAN headers, can also be configured for operational demands.

The dynamic configuration block allows operators to change the parameters of Gatekeeper and Grantor servers and to diagnose runtime issues. For example, operators can update the IP ranges handled by GK blocks, list the ARP and ND tables for network diagnosis, update the enforced policy on Grantor servers, and flush all policy decisions cached at Gatekeeper servers associated with a given destination IP.

2.3.3 Hardware Offloading

DPDK applications are most efficient when they can leverage packet processing optimizations via hardware features on specialized NICs. Gatekeeper uses several of these hardware features, including RSS and multiqueues, filters, and checksum offloading.

RSS and multi-queues. To fully utilize the potential of the modern multi-/manycore CPUs, Gatekeeper uses NICs with multi-queue and *receive-side scaling* (RSS) support. The multi-queue support allows different lcores to access different

queues on the NIC without contending with each other. RSS is used to distribute incoming packets to the different queues, each of which is serviced by a different lcore. RSS ensures that packets belonging to the same flow will be processed by the same lcore in the same order as they arrive at Gatekeeper server, thus avoiding out-of-order packet delivery. A NIC that is RSS-enabled uses a hash function to compute a hash value over a defined area in packet headers. The defined area can be non-contiguous, but Gatekeeper hashes over the IP source and destination addresses. A number of least significant bits of the hash value are used to index an indirection table. The values in the indirection table are used to assign the received data to a queue/lcore. In order to calculate the RSS hash value, the hash function uses a secret key of the RSS hash as input. To mitigate vulnerabilities that target the RSS hash, Gatekeeper randomizes the secret key on startup.

Filters. For best performance, Gatekeeper leverages NICs that support EtherType and ntuple filters. EtherType filters allow packets to be steered on the basis of the EtherType field of the Ethernet header. This allows, for example, ARP packets to be steered to the LLS block without touching the data plane of the Gatekeeper or Grantor servers. Ntuple filters allow packets to be steered to queues on the basis of L3 and L4 headers. This allows, for example, BGP packets to be directed to the CPS block. Without EtherType or ntuple filters, all packets are received by the GK block and are distributed to the other blocks using ACLs to find the correct functional block, and Gatekeeper mailboxes to transfer the packets.

Checksums. Since all packets that Gatekeeper decides to forward are encapsulated, the outer IP header requires checksum computation. To minimize the encapsulation effort, IP header checksum computatoin is offloaded to hardware when supported by the NICs, saving CPU cycles. Similarly, packets sent from Grantor to

Gatekeeper (policy decisions) are sent using UDP, and therefore L4 checksum offloading is also enabled when supported by hardware. When checksum offloading is not supported, Gatekeeper calculates the checksums in software.

2.3.4 Software Techniques

In addition to hardware features, Gatekeeper leverages several optimizations in software to speed up packet processing, including batching, prefetching, lockless rings, and access control lists. A more thorough overview of the techniques we used is described in Fu (2020).

Batching. To reduce system call overhead, Gatekeeper utilizes batching as often as possible. Reads and writes to the network interfaces, transfers of packets to different blocks via mailboxes, access control list lookups, flow lookups, and LPM table lookups are all performed in batches.

Prefetching. Since data plane performance is critical to Gatekeeper, we added prefetching instructions where needed to allow Gatekeeper to fetch data for the L1 cache ahead of demand, hiding memory latency by removing load instructions from the critical path.

Lockless rings. Often, data plane packets or Gatekeeper metadata needs to be passed between functional blocks. For example, request packets are sent to the Solicitor block from the GK block for priority queueing. To efficiently perform this handoff, Gatekeeper uses the lockless consumer/producer ring library of DPDK.

Access control lists. Gatekeeper depends on the ability to steer certain types of control plane packets to the CPS, LLS, and GGU functional blocks. When the available hardware does not support EtherType filters or ntuple filters, or when certain types of packets cannot be adequately matched using those filters, Gate-

keeper uses DPDK's highly performant access control list library to match packets and transmit them to different blocks.

2.3.5 Operational Features

Gatekeeper provides several features that are key for real world deployments. These operational features include bonded devices and LACP, VLAN configuration, routing daemon support, fragmentation handling, and operational logging.

Bonded devices and LACP. To meet operational demands, Gatekeeper enables the Link Aggregation Control Protocol (LACP) to automatically bond physical ports to a single logical channel, which is required by some Internet exchanges.

VLAN. It is common practice for IXPs to assign VLAN IDs to each member, and require each frame to carry a VLAN tag. Gatekeeper supports variable-length Ethernet headers so that VLAN tags may be inserted or removed as desired. Gatekeeper also supports separate VLAN tags for IPv4 traffic and IPv6 traffic.

Control plane using KNI and BIRD. Since Gatekeeper forces its attached network interfaces to use DPDK and forego the Linux kernel networking stack, other standard network tools cannot run on Gatekeeper interfaces. To reduce barriers to deployment, Gatekeeper provides the ability for existing control plane tools (such as routing daemons) to be used as-is alongside Gatekeeper by using the DPDK kernel-NIC interface. This creates a virtual network interface that a routing daemon can run on, while being able to make route announcements and adjust the routing tables of Gatekeeper. We created a patch (Doucette, 2020) of the control plane tool BIRD (BIRD, 2020) to be able to deliver routing table updates directly to Gatekeeper instead of the Linux kernel.

Fragmentation. At Gatekeeper servers, fragmented packets (which often indi-

cate abuse) can be given secondary bandwidth or dropped using policy enforcement programs. Grantor servers may need to reassemble fragmented packets before making policy decisions. To do so, they utilize the DPDK IP fragmentation and reassembly library. Additionally, to avoid attackers overflowing the request channel with fragments that never complete, flows associated with fragments that have to be discarded before being fully assembled are punished.

Log rate-limiting. Gatekeeper provides configurable logging for every functional block and library. Importantly, in a DDoS mitigation system, the logging subsystem may become a target of attack by malicious actors hoping to exhaust Gatekeeper's storage or waste CPU cycles on I/O. Therefore, each functional block defines a parameter to rate limit the number of logs per unit time.

2.4 EVALUATION

2.4.1 Goals

In this section, Gatekeeper is evaluated along several axes:

- **Basic functionality and accuracy.** It should accurately reflect the policies it is given to enforce and sufficiently mitigate infrastructure layer attacks to allow legitimate clients to maintain service. Since the request channel is a critical component of network capability systems, it should also mitigate attacks on the request channel itself. Simply put: does Gatekeeper work as advertised?

- **The effect of different policies.** Policy enforcement programs are more than just simple token bucket algorithms, and have the ability to inspect transport layer headers for the purposes of secondary bandwidth limits, apply negative bandwidth, etc. How does policy richness affect the mitigation attacks?

- **Performance benchmarking.** The peak attack capacity is of particular interest for DDoS mitigation systems. How far can Gatekeeper be pushed against various workloads and load testing parameters?

- **Cost analysis.** Given the cost asymmetry between attackers and defenders in the DDoS game, deployment cost is an important metric in evaluating a mitigation system. We consider both capital and operating expenses in asking: how costly is Gatekeeper?

2.4.2 Testbeds

The Gatekeeper evaluation is performed across two testing environments: (1) a Dell PowerEdge R640 server, which is used for performance benchmarking a Gate-

keeper server; (2) Amazon Web Services Elastic Compute Cloud (AWS EC2), which is used for end-to-end evaluations that include clients, Gatekeeper servers, Grantor servers, and destination servers.

2.4.2.1 PowerEdge Server Specifications

To benchmark a Gatekeeper server in a realistic production environment, we use a Dell PowerEdge R640 rack server. The server has 768 GB of memory along with two Intel Xeon Silver 4214R 2.4 GHz processors, each equipped with 16.5 MB of cache space and 12 cores with 24 threads. For networking, the server has two Intel X550 10 Gbps adapters.

2.4.2.2 AWS EC2

The Amazon testbed provides an environment where Gatekeeper can be tested end-to-end, to measure the effect the system has on legitimate clients during an attack. The testbed is composed of an attacking client instance, which runs a packet generator to launch floods from 16k virtual attackers (source IPs) to a destination Web server in an attempt to overwhelm the destination's available bandwidth. The testbed also has a legitimate client, which uses `curl` to repeatedly upload a 20 KB file to the destination server. For each experiment, the legitimate client uploads the file 50 times, and the average file transfer time is computed across all transfers. For experiments where Gatekeeper is used, packets are redirected from the attacking client and the legitimate client to Gatekeeper using encapsulation. Gatekeeper transmits some subset of these packets (according to the policy for a given experiment) to Grantor, and packets sent along the Gatekeeper-Grantor path are processed by a Linux router instance that enforces the Gatekeeper queueing algo-

rithm. The topology of the testbed is shown in Figure 2.6.

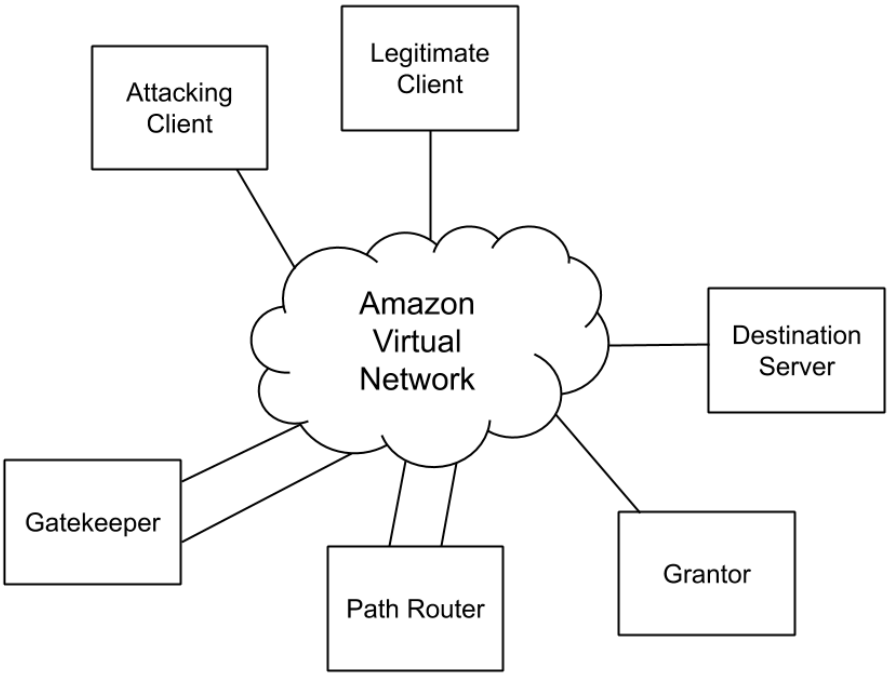


Figure 2.6: Amazon testbed topology.

However, there are some limitations with the Amazon testbed:

- **Instance type.** Amazon provides a range of virtual machine instance types for different use cases and applications. Not all instance types use NICs that DPDK supports. Gatekeeper and Grantor must be run on instance types that use the Elastic Network Adapter (ENA) (Barr, 2016), which is supported by DPDK.

- **Lack of support for some RSS features.** Although the Amazon ENA nominally supports RSS to spread flows to multiple instances of the GK block, we found that in practice its support for various RSS features was lacking, and

therefore could not fully be used with Gatekeeper. Without RSS, the GK and GT blocks cannot scale up to use more lcores, and therefore Gatekeeper's and Grantor's processing power is limited to a single lcore for data plane processing.

- **Lack of hardware offloading.** The ENA does not support some hardware features that Gatekeeper would otherwise leverage, including EtherType filters and ntuple filters. This means that protocols like ARP and BGP, which would otherwise have been directed to the LLS and CPS blocks automatically, take up CPU cycles to be received by the GK block and sent via mailboxes to their appropriate blocks.

- **Low packet throughput.** Users on EC2 can be rate-limited by the policies of the Amazon virtual network. During testing, it was found that with minimum sized packets, 64 bytes, the Amazon virtual network would cap transfer limits at 300 Mbps across instances, even if more instances were added.

- **Inconsistent byte throughput.** Some instances on EC2 are given a variable amount of network capacity, e.g., "up to 5 Gbps," which in practice may only result in 2-3 Gbps on average with a peak performance of 5 Gbps.

- **Bandwidth limits are imposed per-instance.** Bandwidth limitations are applied per-instance, not per-interface. For example, a Gatekeeper server (with both a front and back interface) running on an instance with a 10 Gbps capacity must split that capacity between receiving packets on the front interface and sending them on the back interface.

- **Fair queueing.** From our experiments, we speculate that there may be some type of per-flow fair queueing imposed by the Amazon virtual network.

When the attacks are split between a relatively small number of attacking flows (1K-2K), the legitimate client is always able to immediately connect, regardless of overall attack strength. However, significantly increasing the number of flows (16k) seems to nullify this effect, causing the legitimate client to wait proportionally longer as the attack strength increases.

The effect of these limitations is that Gatekeeper cannot be consistently tested at the scale of tens of Gbps with minimum packet size, which is a realistic and desirable evaluation scenario, since clients cannot consistently achieve rates that high, and Gatekeepers, Grantors, and destination servers can not consistently receive and transmit at rates that high. Moreover, because of the limitations of RSS and other hardware features, Gatekeeper and Grantor are limited in their packet processing capacity.

Therefore, the EC2 instance type for the destination server is chosen to have fairly low capacity, around 1 Gbps, and the system is evaluated using only up to around 10 Gbps of attack traffic composed of large (1024 byte) packets. Unless otherwise noted, the attacks use floods of TCP traffic, i.e., TCP packets carrying application data, without SYNs. We use a static set of 16,000 attacking flows regardless of the attack strength. Unless otherwise noted, Gatekeeper servers use the m5.8xlarge instance type.

2.4.3 Baseline Functionality

To evaluate Gatekeeper at its most basic functionality, we measured its effect on the file transfer time of a legitimate client during a flooding attack. To do so, we used the Amazon EC2 testbed to flood a destination server with up to 10 Gbps of traffic from 16,000 flows. The traffic was first directed through Gatekeeper, which

granted all flows using the baseline granted BPF program (no secondary or negative bandwidth) at rates of 16 Kbps, 32 Kbps, and 64 Kbps.

During the attack, we used a legitimate client to upload a 20 KB file to the destination server using the `curl` application. The results are shown in Figure 2.7.

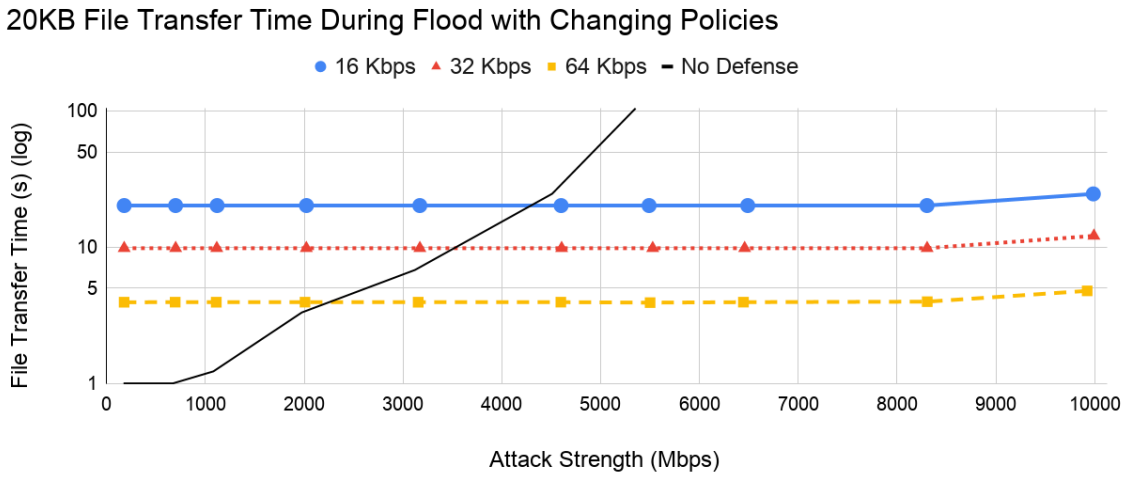


Figure 2.7: The time to transfer a 20 KB file during an attack while varying the policy's granted rate limit.

The result is that the rate limit of the granted flows has much more of an impact on the legitimate client's file transfer time than the attack does. In fact, the attack does not seem to have an effect, except at around 10 Gbps, which is the capacity of the Gatekeeper server. At very low rates, such as 16 Kbps, it takes about 20 seconds (20 token bucket refill periods) for the legitimate client to be able to transfer the file. As the rate limit increases to 32 Kbps and then 64 Kbps, the legitimate client is able to transfer the file more quickly while the attack itself is still mitigated.

For reference, we also included trials of the legitimate client trying to connect when no Gatekeeper or Grantor servers are running. The file transfer time increases exponentially up through trials with 5 Gbps of attack traffic. Beyond that

point, the legitimate client times out when trying to perform the transfer.

2.4.4 Effect of Policies

We now further explore the effect that policies can have by introducing the secondary bandwidth and negative bandwidth mechanisms. More information and example policies can be found in Section 3.4.

2.4.4.1 Secondary Bandwidth Limit

A secondary bandwidth limit (described further in Section 3.4.3.1) allows Gatekeeper to impose a second, lower limit for certain types of traffic within flows, such as ICMP, UDP, or TCP SYN packets. In this evaluation, we change the traffic flood to be composed of TCP SYN packets, and change the policy decision to apply a secondary bandwidth to TCP SYNs. The primary bandwidth limit is 256 Kbps, and the secondary bandwidth is set to be 1.6 Kbps (5% of the primary limit). The results are shown in Figure 2.8.

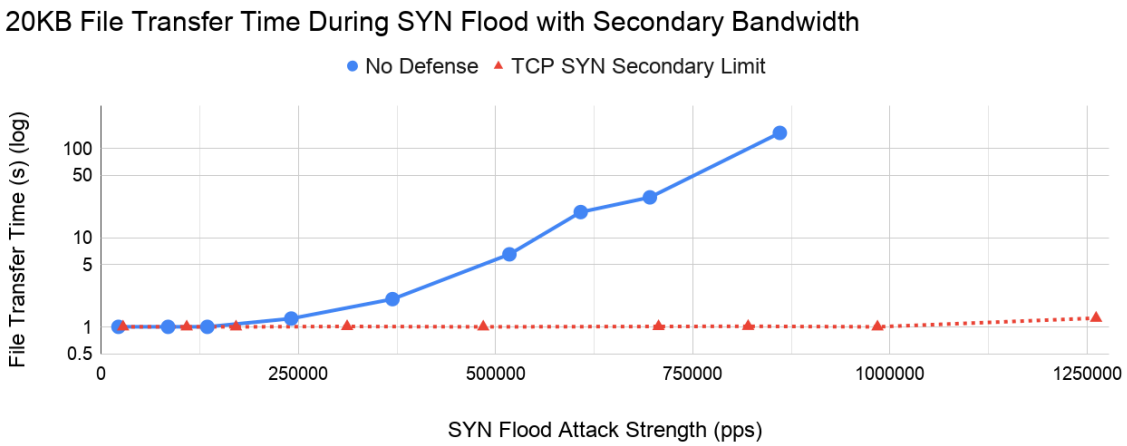


Figure 2.8: The time to transfer a 20 KB file during a SYN flood attack with no defense and with a Gatekeeper policy that bins TCP SYNs into a secondary bandwidth limit.

The result is that the TCP SYN flood is instantly mitigated by Gatekeeper. Up through around 1.25 Mpps, Gatekeeper sufficiently rate limits the TCP SYNs so that the legitimate client is able to connect and transfer the file in a single rate limiting period.

2.4.4.2 Negative Bandwidth

Policy enforcement programs that use negative bandwidth (described further in Section 3.4.3.2) can punish flows that transmit beyond their allotted limit. We evaluate the effect of negative bandwidth by building off of the basic policy evaluation, increasing the rate limit for each flow to be 128 Kbps and then 256 Kbps during attacks of increasing strength. We then performed trials with the same rate limits, but applied a policy that enforced negative bandwidth when flows exceeded those limits. For all trials, we measured the average file transfer time of a legitimate client. The results are shown in Figure 2.9. tically

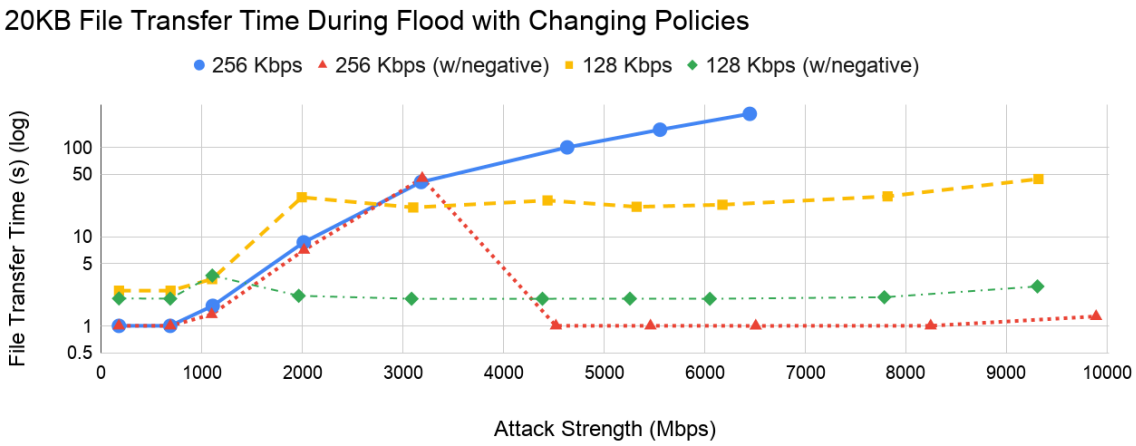


Figure 2.9: The time to transfer a 20 KB file during an attack with a policy that applies negative bandwidth.

The result is that negative bandwidth policies can have a major impact on al-

lowing legitimate clients to maintain service. Between 1 Gbps and 2 Gbps, attacking flows that are allotted 128 Kbps need to send above that limit to keep up the attack strength. By 2 Gbps, it takes 10s for the file to be transferred, although Gatekeeper limits the attack traffic sufficiently all the way up to 10 Gbps to keep the file transfer time at 10s. On the other hand, the 128 Kbps policy with the negative bandwidth ("w/negative") forces the attackers to push their bandwidth allotment increasingly negative as the attack strength increases. By 2 Gbps, the legitimate client is able to transfer the file as if there were no attack at all.

The same outcome is observed for the 256 Kbps trials. Although the higher limit allows the attack to be successful at higher rates, driving the file transfer time to be 50s by 3 Gbps, soon after that the bandwidth allotment is overwhelmed and the legitimate client is able to quickly transfer the file.

2.4.5 Performance Benchmarking

We also measured the performance of a single Gatekeeper server when under stress by evaluating it against an attack with high source address churn on the PowerEdge server. We fully randomized the source address of the attacking flows, creating an indefinite stream of new flows, aiming to exhaust the flow table state in GK blocks. We also used attack traffic composed of just 64 byte packets. On handling packets, Gatekeeper performed the simplest possible action: a drop. We measured the capacity of Gatekeeper to process this traffic by varying the flow table size[2] and the number of GK instances (lcores/threads). The results are shown in Figure 2.10.

[2]Measured in flow table entries, each of which is 128 bytes.

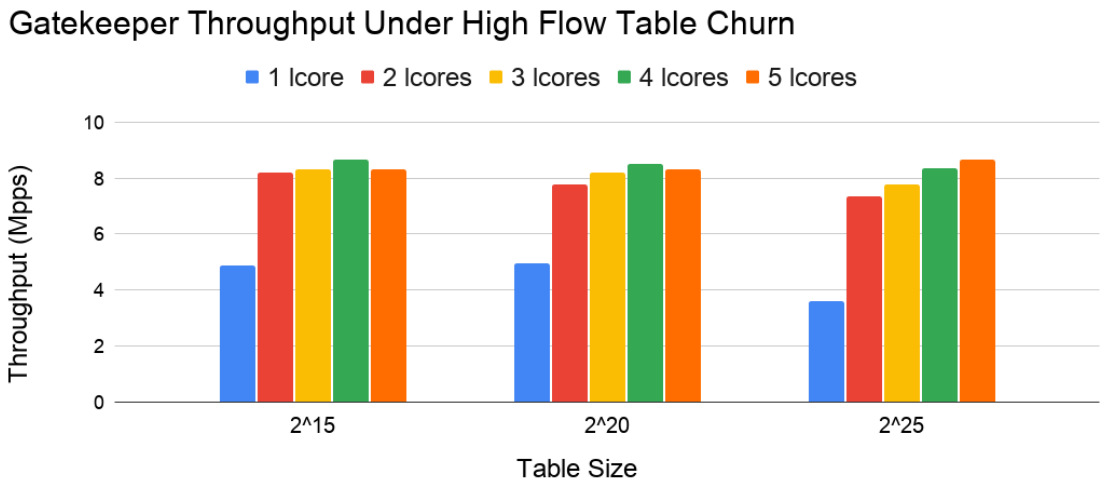


Figure 2.10: The throughput of Gatekeeper during an attack with maximum flow table churn and minimum packet size.

The result is that even under extreme conditions – minimum sized packets, a full flow table, and new flows constantly arriving – Gatekeeper can process at least 8 Mpps (up to the packet generator's maximum limit) under certain flow table and GK instance configurations. For context, being able to process 10 Mpps is considered highly performant (Majkowski, 2018). Additionally, 8 Mpps is the equivalent of 5 Gbps when using minimum-sized packets.

There is a significant benefit in increasing the number of GK block instances/lcores from one to two, as the throughput nearly doubles. Beyond two GK instances, the benefit may appear insignificant, but this is likely because Gatekeeper is already processing packets as fast as the generator is sending them. Therefore, measuring the effect of more GK instances may require separating the packet generator onto its own hardware.

2.4.6 Cost Analysis

One of the motivating factors that drove the design of Gatekeeper was to keep it affordable and within reach of smaller ISPs, enterprises, institutions, etc. In this section, we provide a back-of-the-envelope evaluation of the cost of Gatekeeper, as well as an evaluation regarding the effectiveness of Gatekeeper deployments in Amazon, using instances at different prices.

2.4.6.1 Deployment Cost

We now present rough estimates of the operational cost of two Gatekeeper deployment scenarios: (1) a deployment of 2.3 Tbps of bandwidth protection, which represents one of the largest reported DDoS attacks by volume (AWS, 2020a), as well as (2) a deployment that balances peak capacity and cost. We contrast our cost estimates with defense costs that were made public during the headline-grabbing DDoS attack against *KrebsOnSecurity*, a popular Internet security blog (Krebs, 2016).

To shield a single AS with 2.3 Tbps of bandwidth protection, our modeled deployment would use 23 VPs, each with 100 Gbps of incoming traffic capacity. Conservatively estimating the cost of operating at each VP at \$5K per month[3], the operational cost of this deployment would be \$1.38M per year. This estimate includes the quoted price to contract a link in an IXP, and then is rounded up to guesstimate additional operational costs, such as those to contract layer 2 connectivity to the deploying network. While \$1.38M per year is out of reach for many deploying ASes, it is favorable compared to the cost of "millions of dollars" that Akamai estimated (Bray, 2016) was needed to protect *KrebsOnSecurity*.

[3]We could not identify public data to back this estimate, but our industry partners have verified that this value covers their market price estimate to deploy Gatekeeper at an IXP.

While rather large DDoS attacks do grab headlines, according to Arbor Networks, 99% of the DDoS attacks in 2016 peaked at less than 20 Gbps (Anstee et al., 2017, Figure AT3). Thus, an AS willing to endure downtime when a 1% tail DDoS attack hits would have a much cheaper deployment: $12K per year, using the same assumptions as the first estimate. Not only is this operational cost in reach of small and medium sized potential deployers, but it also stands favorably against a cheap 20 Gbps-protection contract against infrastructure-layer attacks offered by an established security company[4]: $24K per year.

Due to the fact that small and medium sized deployers may lack the administrative resources to manage a deployment, they may not be able to deploy Gatekeeper on their own. But, for the cost of $12K per year, 150 such ASes could afford the 2.3 Tbps protection described above, and would still have $420K per year to share professional management of the system. According to the AWS Shield pricing (AWS, 2020a), the yearly fee alone for 150 companies would be $5.4M. Taking the calculation one step further, a provider like AWS likely has more than 1,000 client members, pushing the total cost to approximately $36M per year. With this sum, Gatekeeper could be operated at 7,200 VPs.

The estimated costs of the Gatekeeper deployments above show that Gatekeeper is cheaper and scales better than the current solutions in the market, potentially reducing cost by more than 3X. Finally, note that this cost analysis is conservative, since all deployment scenarios are based on IXPs. Deployments that use cloud providers may further lower the operational cost, since cloud providers enable ASes to quickly scale up and down the number of VPs and computing capacity without infrastructure investment.

[4]We are not at liberty to publicly state which company.

2.4.6.2 *Amazon Instance Variation*

We also evaluated how changing the instance type (and therefore price) of the Gatekeeper server in the Amazon testbed affects the results of some of the previous experiments. For all of the previous Amazon experiments, our baseline instance type for the Gatekeeper server was the m5.8xlarge. This is a fairly expensive instance type, and there are cheaper instances in the same family that could be used, trading off CPU and memory for cost. We compared three instance types in the same family in the US East (Ohio) region, as shown in Table 2.2. For this region, the savings between the instance types scales linearly. The cost savings per hour, compared to using the m5.8xlarge instance, is also shown.

Instance	vCPUs	Mem (GiB)	B/W	Price/Hr	Savings/Hr
m5.8xlarge	32	128	10 Gbps	$1.536	N/A
m5.4xlarge	16	64	Up to 10 Gbps	$0.768	$0.768
m5.2xlarge	8	32	Up to 10 Gbps	$0.384	$1.152

Table 2.2: Different types of instances evaluated for a Gatekeeper server in AWS EC2.

Since Gatekeeper requires at least 5 vCPUs for basic operation (one for each of the relevant functional blocks), cheaper instances with fewer vCPUs than the m5.2xlarge could not be used. The M5 instance type family is for general-purpose computing; other instance type families had significantly poorer network performance for approximately the same price, so only the M5 was evaluated.

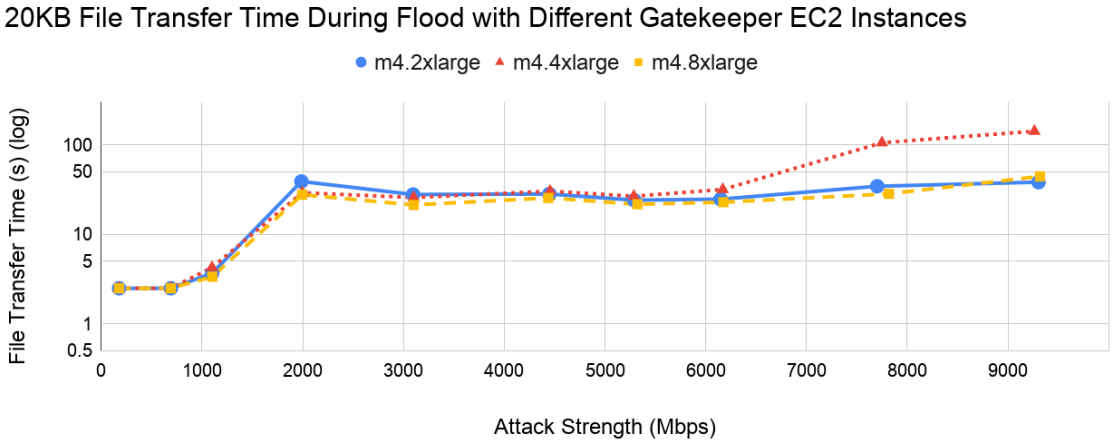


Figure 2.11: The time to transfer a 20 KB file during an attack with varying Gatekeeper instance types and prices.

Figure 2.11 depicts the results of a 20 KB file transfer with attack traffic and Gatekeeper defenses deployed at various AWS instance types. We apply a 128 Kbps rate limit to all 16,000 attack flows as well as the legitimate client flow. We found that for these parameters, there are no significant performance benefits, and approximately the same performance for attacks up to 10 Gbps could be achieved using a m5.2xlarge instance at 25% of the price. However, it is worth noting that if the Amazon testbed fully supported RSS, instances with more vCPUs could utilize more GK instances, which could in theory provide better performance.

CHAPTER 3
Policy Toolkit

3.1 OVERVIEW

Since network capability systems perform admission control for the network layer connections into the destination network, the network operator must have mechanisms to decide which flows should be granted access and at what rates. Most previous capability proposals only provide a high level description of how such destination policies would work, allowing access only in response to outgoing traffic or to registered users (Anderson et al., 2004), rate limiting all traffic according to link load and packet loss measurements (Liu et al., 2010), or by using default byte rates or application-layer cookies and CAPTCHAs (Yang et al., 2005). Still other capability proposals do not discuss the destination policy issue, focusing only on the capability mechanism (Yaar et al., 2004).

In order to be deployable, Gatekeeper provides practical mechanisms for destination policy specification. In this chapter, we provide a policy toolkit as a guide for network operators looking to create a Gatekeeper deployment. Gatekeeper actually has two policy mechanisms: one at Grantor servers to make policy decisions about flows, and one at Gatekeeper servers to enforce policies over individual packets. We give an overview of both mechanisms in Section 3.2, and describe how to map the specifications of a network under deployment to a set of policies in Section 3.3.

The correctness and precision of the policies can make or break a Gatekeeper deployment, since adversaries that find vulnerabilities in, or take advantage of, policies can circumvent the protections that the system provides. Fortunately,

Gatekeeper provides a rich array of possibilities for applying policies to traffic, which we break down into basic policy techniques that most deployments should use (Section 3.4), as well as advanced techniques that could offer solutions for attack detection, source spoofing prevention, and authentication, as well as provide opportunities for further research (Section 3.5).

3.2 POLICY DESIGN

This section presents the design choices that shaped the format of policies and a deployment-tested template to write policies.

The design of Gatekeeper policies borrows heavily from two streams of prior work: capability and filtering systems to regulate the transmission rate of flows, and SDN to centralize this decision-making process (see Section 2.2.3). However, the design choice of implementing destination policies as *programs* represents a breakthrough in the context of DDoS mitigation systems. Prior work implemented policies as sets of rules that are pattern matched to packets or flows. These rules and patterns are described under a predefined declarative language. The motivation for using declarative languages is to provide abstractions for hardware operations and filters. These languages, however, become the weakest link in a DDoS protection system when attacks target their limitations. For example, the Catch-22 attack shows that attackers can force source address filtering mechanisms in legacy routers into an untenable position: either use coarse-grained (/16 prefix) filtering rules and incur large amounts of collateral damage, or use fine-grained filtering rules (/32 prefix) and risk not mitigating the attack (Shi et al., 2019).

Running Gatekeeper policies as programs does not require giving up the hardware abstractions. However, it enables us to move away from declarative languages and toward bytecode virtual machines (VMs). We chose to adopt policies as programs in two ways:

1. A set of policy *enforcement* programs that are run at Gatekeeper servers. On ingress, a packet's flow is extracted and mapped to the executable enforcement program that it has been assigned. Policy enforcement programs may keep state, e.g., for a token bucket algorithm, to decide whether to drop or

accept traffic at a prescribed rate.

2. A single policy *decision* program that is run at Grantor servers. The decision program maps flows to policy enforcement programs, and installs rules reflecting those decisions at Gatekeeper so that the enforcement program can be run on subsequent packets in the given flow.

Breaking the policy process up into decision-making and enforcement parts has two advantages. First, it enables Gatekeeper to take prompt action on flows that misbehave *after* receiving a favorable policy decision, e.g., by applying secondary rate limits or by tracking negative bandwidth at policy enforcement time (Section 3.4). Previous capability systems have relied on capability expiration or bandwidth caps to mitigate this issue (Yaar et al., 2004; Yang et al., 2005).

Second, it allows us to separately choose the best fitting bytecode VMs for Gatekeeper and Grantor servers. Gatekeeper servers perform policy enforcement using VMs that run extended Berkeley Packet Filter (eBPF) programs (McCanne & Jacobson, 1993), whereas the policy decision program running on Grantor servers are implemented using a Lua VM (Ierusalimschy et al., 2006). The key requirements that led us to choose Lua on Grantor servers was (1) its support for dynamically editing the policy (e.g., redefining functions, modules, variables) in order to enable changing the policy with minimum or no impact to deployed systems, and (2) ease of integration with external libraries (typically in C). eBPF was chosen to implement policy enforcement programs due to (1) the availability of static analysis to guarantee termination, memory safety, and bounded resources (Gershuni et al., 2019) and (2) ease of packet inspection.

While eBPF has been used in production to implement packet redirection in load balancers (Wragg, 2020; Shirokov & Dasineni, 2018) and high-performance,

ad hoc packet filters in DDoS protection systems (Fabre, 2019), the use of eBPF programs in Gatekeeper to perform policy enforcement brings greater flexibility. The difference in these approaches is in the amount of state space under the control of eBPF programs. For example, eBPF programs in Gatekeeper can limit bandwidth per flow, while the eBPF programs running on other systems have only enough state to do so for a limited number of classes.

3.2.1 Decision Types

Each decision made by a policy decision program falls into one of three categories: declined, granted, or BPF.

For declined flows, the only parameter that needs to be specified by the decision program is the lifespan of the decision. Although declined decisions can be used when attacks are detected, defining an attack is widely known to be a difficult problem in general. Therefore, we recommend mostly using declined decisions for malformed packets or well known abusers such as bogons and malware (Section 3.4.1). A reasonable timeout is on the scale of minutes, so that the effects of false positives (e.g., malformed packets due to misconfiguration) and collateral damage (e.g., innocent users who are behind the same source NAT process as abusive users) are lessened, with the option for cheaply extending the declined decision on expiry.

For granted flows, there are four parameters: the rate limit of the capability as well as three arguments that define when the capability expires and how the capability renewal request may proceed. The setting of these parameters is highly dependent on the network profile mapping.

For BPF flows, there are four parameters: the lifespan of the decision, the index

of the installed BPF program to be used at Gatekeeper, the length of a cookie to be associated with the program, and the cookie itself. The cookie is a piece of memory passed to the BPF program at execution time, allowing Gatekeeper to keep arbitrary per-flow state up to 64 bytes in size. Note the distinction between granted flows and BPF flows: the built-in granted channel mechanism in Gatekeeper simply applies a single bandwidth limit to the flow, whereas BPF programs are also generally applied to granted flows, but can flexibly inspect packet headers, keep state, enforce multiple bandwidth limits, etc.

We next describe how to map the characteristics and requirements of a deploying network to policy decision and enforcement programs.

3.3 WRITING POLICIES

We now present guidelines for writing policies in a Gatekeeper deployment. In our discussions with network operators, we found that a policy is most naturally organized as follows: policy decision programs (in Lua) inspect IP addresses to map flows to policy enforcement (eBPF) programs, and eBPF programs inspect transport headers to monitor the behavior of flows and adhere as best as possible to the given policy decision.

As a rule of thumb, each eBPF program reflects a network service *profile*. For example, consider the profile of outgoing email servers. They have no listening sockets – they only open connections to the SMTP port of remote email servers, and these connections have very small ingress traffic footprints. This profile-to-program heuristic leads to a simple breakdown of the work to write a policy: (1) identify all network profiles, (2) write an eBPF program for each of those profiles, and (3) map flows to those eBPF programs in the Lua policy.

Following this proposed breakdown of work, Lua policies are given a number of network prefixes with which to classify incoming packets based on their destination addresses. We use longest prefix matching on destination addresses by default, but also adopt flexible address classification to enable approximate source location (MaxMind, 2020), threat identification (Team Cymru, 2020; The Spamhaus Project SLU, 2020), and to check the purpose of the source (Cloudflare, 2020b; Amazon Web Services, 2020; Google, 2020b), etc. This classification of source addresses can be used to decide on denying or granting communication, limiting bandwidth, and differentiating service.

While translating a network profile into an eBPF program, the policy writer should classify packets into three bins: primary, secondary, and unwanted traffic.

Primary traffic carries the main purpose of the service, while secondary traffic is permitted traffic that has no reason to be present at the same scale of the primary traffic. Examples of secondary traffic are TCP SYN, ICMP, and fragmented packets. Once the code for the classification of packets is in place, the policy writer overlays it with two bandwidth limits: one limit before the classification to control the bandwidth of the flow as a whole, and another limit for secondary traffic after the classification. These limits can be implemented as token bucket algorithms.

Policies that follow the template explained above deal with the most common forms of infrastructure attacks such as floods (e.g., SYN, UDP, ICMP), amplifications (e.g., DNS, NTP, Memcached), and arbitrary combinations of these attacks (also known as multi-vector attacks). The fraction of the attack traffic that bypasses these policies depends on how narrow the network profiles are, how precise they are implemented in the policy enforcement programs, and the quality of the classifications of the source addresses in the policy decision program. But even if these factors are adjusted to perfection, some traffic attacks can only be identified by the protected applications or intrusion detection systems (IDSes). In this case, applications and IDSes can feed the policy decision program with source addresses and packet signatures to mitigate this sneaky traffic through ad hoc policy enforcement programs and assignment of lower bandwidth limits. Gatekeeper cannot help when there is no observable distinction between attack and legitimate traffic.

Finally, the policy template presented in this section leaves room for a potential avenue for improvement: the fact that Grantor servers are colocated geographically. This geographical proximity often translates into low latency and ample bandwidth between Grantor servers, which in turn, enables the employment of a distributed database for the use of policies. With the help of this database, Lua

policies could potentially identify spoofed source addresses analyzing the (source address, incoming vantage point address) pair for inconsistencies, as well as make sophisticated bandwidth allocation based on the source AS and load of the links behind the Gatekeeper servers.

3.4 BASIC POLICY TECHNIQUES

Recall that for each request (or capability renewal) packet received by Grantor, the GT block parses it and passes a structure of packet information to the Lua policy, which maps the packet data to a policy decision. To help accomplish this task, Gatekeeper provides a basic set of policy functionality that will likely be used by most deployments, which includes support for performing longest prefix matching, port lookups, secondary and negative bandwidth, and the ability to rapidly support new protocols as needed.

3.4.1 Host Lookups and Bogons

As described by the heuristics for writing policies, decisions will mainly be driven by information derived from the network layer header, especially the source and destination IP addresses.

Although source addresses can be spoofed (an issue that is addressed in Section 3.5.1), there are simple checks that can be performed to filter flows based on invalid addresses, or *bogons* (Team Cymru, 2020). Bogons include any IP address in a range not allocated by IANA, or any IP address in a range reserved for private or special usage. Actively maintained and publicly available bogon lists can be included in a Gatekeeper policy decision program. Lists of invalid prefixes are imported when the policy decision program is loaded, and they are then inserted into a longest prefix matching (LPM) table. When requests arrive, the policy decision program extracts the source address, performs a lookup against the bogon LPM table, and if there is a match, the flow is declined. Source addresses can also be checked against per-AS reputation mechanisms (Konte et al., 2015) and reputation databases that identify known sources of cyberthreats (NERD, 2020).

Destination IP addresses can similarly be used as the basis of a policy decision. If there are no indications of abuse based on the source, a decision can be made according to the characteristics of the protected host in the destination network. Dedicated Web servers have much different characteristics than dedicated mail servers, so network operators may find it useful to map certain hosts (destination IP addresses) to policies on this basis, adjusting the capability's rate limit and expiration time as appropriate. Clusters of hosts that can easily be represented by a prefix can be inserted into an LPM table for more efficient lookups. A typical deployment may consist of dozens of such host-to-policy mappings.

An example of a policy that primarily uses source and destination IP addresses as the basis for a decision is shown in Listing 3.1. The program looks up the source IP address in the LPM table (which has been populated with bogon prefixes), and if a result is found, the flow is declined. If the source address passes the check, the destination IP is checked against two static IP addresses that represent mail and Web servers, and although both are given granted decisions, the rate limit and capability expiration parameters differ slightly. All non-IPv4 traffic is declined. Programs also have the ability to base policy decisions on other IP header fields, such as the Gatekeeper priority, the ECN bits (Floyd et al., 2001), or whether the packet is fragmented. The policy example applies a declined decision with a longer-than-usual expiration time as a punishment for fragmented packets, which are often abusive.

Note that some code details that are extraneous to our purposes here have been stripped away or simplified, such as endian conversion, library names, structure names, etc. A syntactically correct example can be found in the Gatekeeper source code (Machado et al., 2020).

```
-- Static addresses of two protected hosts.
mail_server, _ = str_to_prefix("192.0.2.1/32")
web_server, _ = str_to_prefix("192.0.2.2/32")

-- IPv4 bogon list.
for line in io.lines("bogons-ipv4.txt") do
    local ip_addr, prefix_len = str_to_prefix(line)
    lpm_add(ip_addr, prefix_len, DECLINED)
end

function policy_lookup(pkt_info, decision)

    if pkt_info.frag then return declined(decision, 600) end

    if pkt_info.ip_ver == IPV4 then
        local ipv4_hdr = (struct ipv4_hdr *)pkt_info.l3_hdr

        decision = lpm_lookup(ipv4_hdr.src_addr)
        if decision ~= nil then
            -- Decision from source address LPM lookup.
            return decision
        end

        -- Make decision based on host/service.

        if ipv4_hdr.dst_addr == mail_server then
            return granted(decision,
                64,                   -- TX rate
                3600, 3000000, 15000) -- decision expiration
        end

        if ipv4_hdr.dst_addr == web_server then
            return granted(decision,
                512,                -- TX rate
                600, 540000, 3000) -- decision expiration
        end

        -- Destination IP not recognized.
        return declined(decision, 60)
    end

    -- Decline all non-IPv4 flows.
    return declined(decision, 60)
```

Listing 3.1: A policy decision program in Lua that maps source and destination IP addresses to decisions.

3.4.2 Port Lookups

As another line of defense, the policy writer may find it useful to define policies based on port numbers, which provides finer granularity for making decisions, especially for servers that host multiple services. For example, a server that hosts both encrypted (HTTPS) and non-encrypted (HTTP) Web services can grant different rate limits on a per-port basis. It also allows an operator to block flows that are performing network reconnaissance, i.e. probing the destination network to see which hosts and ports are available to try to find vulnerabilities.

An example of a simple policy decision program that maps ports to decisions is shown in Listing 3.2. There are several elements to this policy, including:

- A lookup table that maps (IP version, destination port) combinations to policy decisions. For IPv4, a flow that is trying to access port 23 (Telnet) results in a declined decision, whereas port 80 (HTTP) is granted for both IPv4 and IPv6. All other ports result in a declined decision.

- A validity check that makes sure the TCP packet is not malformed, based on its length. Malformed packets result in a declined decision for the flow.

- A declined decision for any frame that is not a TCP packet.

```
function default_granted(decision)
    return granted(decision,
        512,                -- TX rate
        600, 540000, 3000) -- decision expiration
end

function default_declined(decision)
    return declined(decision, 60)
end

local simple_policy = {
    [IPV4] = {
        [23] = default_declined,
        [80] = default_granted,
    },
    [IPV6] = {
        [80] = default_granted,
    },
}

function policy_lookup(pkt_info, decision)

    local l3_policy = simple_policy[pkt_info.ip_ver]
    if pkt_info.l4_proto == TCP then
        if pkt_info.upper_len < sizeof(struct tcp_hdr) then
            return default_declined   -- Malformed packet.
        end

        local tcphdr = (struct tcp_hdr *)pkt_info.l4_hdr)
        local decision = l3_policy[tcphdr.dst_port]
        if decision == nil then
            return default_declined  -- Port not found.
        end

        return decision   -- Port found.
    end

    return default_declined   -- Not a TCP packet.
```

Listing 3.2: A policy decision program in Lua that maps ports to decisions.

Since network profiles typically yield a set of valid ports for a deployment, mapping ports to policy decisions can be a quick and simple way for operators to define admission and rates based on application expectations, as well as decline flows that are attempting to access invalid ports. However, since policy decisions are made in response to requests, which represent the first packet in a flow[1], an attacker could craft a legitimate looking packet to gain a capability, only to then use abusive traffic to launch an attack. Gatekeeper can deal with such malicious behavior using secondary and negative bandwidth.

3.4.3 Secondary and Negative Bandwidth

Using techniques that are more sophisticated than standard admission and rate limiting, such as applying secondary bandwidth limits and keeping track of bandwidth debt for flows, requires using the *BPF* decision type. In the examples shown thus far, when a flow is granted we have not differentiated between a *Granted* decision type and a *BPF* decision type. The difference is that flows that are given a *Granted* decision type use the forwarding and rate limiting mechanism built into Gatekeeper servers, whereas a *BPF* decision type runs a BPF program that decides whether and how to rate limit the flow. A simple example of such a BPF program is to mimic the behavior of the built-in granted packet mechanism by rate limiting the flow under a single bandwidth limit. In Listing 3.1 and Listing 3.2, the decision could equivalently be either of these types.

However, BPF affords us the flexibility of doing further packet inspection to decide how to rate limit flows. Whereas the policy decision program at Grantor is only applied to the first packet in a flow (or upon renewal), the BPF program is

[1]Except in the case of capability renewal requests.

applied to *every* subsequent packet in the flow. This gives Gatekeeper the opportunity to monitor flows even after a capability has been granted.

Note that evaluations for the effect of secondary and negative bandwidth are available in Section 2.4; here, we provide more details about how to encode these mechanisms into the policy framework.

3.4.3.1 Secondary Bandwidth

In addition to a main bandwidth limit that is applied to "normal" traffic, a BPF program can be used to apply a *secondary* bandwidth limit to certain classes of traffic. For example, a Web server would expect a majority of incoming traffic to be over TCP. Therefore, TCP traffic in general can be rate limited at a higher limit, while all other transport layer traffic (UDP, ICMP, ICMPv6) can be given a lower, secondary limit that is considered more appropriate for their expected use.

The secondary bandwidth limit can also be used to stem certain classes of transport and application layer attacks. For example, DNS queries could be assigned the secondary limit based on its well-known destination port number, and therefore attempts to launch DNS amplification attacks will be mitigated instantly at the Gatekeeper server by virtue of the secondary rate limiting, even as other traffic continues as normal. This is ideal for situations where some small usage of a particular application or protocol is acceptable, but overuse indicates abuse. For example, TCP SYN packets need to generally be accepted, but can be slowed to the secondary limit, immediately mitigating TCP synflood attacks.

Listing 3.3 shows an example BPF program that uses a secondary bandwidth limit. As with the Lua programs, simplifications have been made to the BPF code for its use here. See the source code for a full example (Machado et al., 2020).

```
web_enforcement(struct bpf_context *ctx)
{
  struct packet *pkt = ctx_to_pkt(ctx);
  if (primary_limit(ctx, pkt) == DROP_PKT)
    return DROP_PKT; /* Primary budget exceeded. */

  switch (ctx->l4_proto) {
  case IPPROTO_ICMP:
    goto secondary_budget;
  case IPPROTO_TCP:
    break;
  default:
    return DROP_PKT;
  }

  /* Everything below is TCP. */

  if (ctx->fragmented)
    goto secondary_budget;

  struct tcp_hdr *tcp_hdr = ...; /* Extract TCP. */

  switch (tcp_hdr->dport) {
  case 21:  /* FTP command */
  case 80:  /* HTTP */
  case 443: /* HTTPS */
  case 22:  /* SSH */
    if (tcp_hdr->syn) {
      if (tcp_hdr->ack)
        return DROP_PKT; /* Amplification attack. */
      goto secondary_budget; /* Contain SYN floods. */
    }
    break;
  default:
    /* Accept connections originated from our web server. */
    if (tcp_hdr->syn && !tcp_hdr->ack)
      return DROP_PKT;
    /* Authorized external services. */
    switch (tcp_hdr->sport) {
    case 80:  /* HTTP  */
    case 443: /* HTTPS */
      break;
    default:
      return DROP_PKT;
    }
  }
  goto forward;

secondary_budget:
  if (granted_pkt_test_2nd_limit(pkt->pkt_len) == false)
      return DROP_PKT;
forward:
  return FWD_PKT;
}
```

Listing 3.3: A policy enforcement program that uses secondary bandwidth to further rate limit certain classes of transport and application layer traffic.

This BPF program has several noteworthy elements:

- The primary limit is generally applied to TCP traffic and the secondary limit is applied to ICMP traffic. Any other type of traffic is dropped.
- Fragmented traffic is given the secondary limit.
- TCP SYN ACKs sent to protected services are dropped to prevent amplification attacks.
- TCP SYNs sent to protected services are given the secondary limit to contain SYN flood attacks.
- The program doesn't accept TCP SYN packets to any port except those it explicitly knows and approves (FTP, HTTP, HTTPS, SSH).
- The program authorizes traffic from clients to the known services at the primary limit.
- The program authorizes traffic coming *from* Web servers at the primary limit.

This program is an exemplar of the richness that BPF program decisions afford Gatekeeper, and helps mitigate many types of attacks that can be fingerprinted by simply inspecting transport layer headers; see Section 2.4.4.1 for a quantitative evaluation of how such a secondary bandwidth limit can mitigate a TCP SYN flood attack. However, attackers can still abuse a granted capability by crafting packets that fall under the primary bandwidth and flooding the destination network with them. To combat this, BPF programs can also use negative bandwidth.

3.4.3.2 *Negative Bandwidth*

In the classical token bucket implementation for rate limiting, when the bucket does not have a sufficient number of tokens to forward a packet, the packet is dropped and no tokens are removed from the bucket. However, the algorithm can be modified to deduct a number of tokens equal to the packet length even when there is an insufficient number of tokens and the packet is dropped. This allows the number of tokens for the flow to become negative, forcing the source to wait in order for packets to be forwarded. If the source continues to send faster than the bucket refill rate, packets in the flow will be indefinitely dropped. When implemented as a part of a Gatekeeper policy enforcement program, this modification has the effect of punishing flows that try to abuse the granted channel after receiving a capability. The effect of this can be quite dramatic when under a high rate of attack; see Section 2.4.4.2 for a quantitative evaluation of this mechanism.

Of course, even with mechanisms like secondary and negative bandwidth, malicious actors will try to find new vulnerabilities and attack vectors. Applications and transport technologies can emerge and evolve (and so can network architectures (Han et al., 2012; Machado et al., 2015)), and it is therefore imperative that Gatekeeper supports the rapid development of new policy programs.

3.4.4 New Protocol Support: QUIC

Unlike many attack detection and firewall tools, which can require a non-trivial development, review, and publishing process to add new features, Gatekeeper can quickly incorporate changes to policies using the flexibility of programs.

Agility in altering policy decision programs or policy enforcement programs may be needed when new types of attacks are discovered, or as new protocols or

applications emerge. At present, there is ongoing shift in transport mechanisms in the Internet, as the QUIC protocol is deployed as a replacement for the traditional HTTPS stack (Langley et al., 2017). Developed by Google to improve the performance of HTTPS traffic, it was deployed on much of Google's infrastructure by early 2017, including being rolled out for all Chrome and YouTube users. In 2017, the QUIC authors estimated that 7% of global traffic was running over QUIC, and this proportion has likely increased in the years since, as both Facebook (Iyengar, 2018) and Uber (Mahindra et al., 2019) have adopted QUIC for their services. Additionally, deployment will probably continue to grow, as the HTTP/3 specification decrees that the latest generation of Web traffic will utilize QUIC.

Often, new protocols yield new DDoS vulnerabilities, and QUIC is no exception. Its IETF draft specification identifies multiple vulnerabilities, including: (1) amplification attacks, since a relatively small QUIC client hello message triggers a large server response, which includes a TLS certificate in order to quickly perform the TLS handshake; (2) Slowloris attacks, which attempt to keep many connections open indefinitely using a minimum amount of activity; (3) stream commitment attacks, analogous to TCP SYN floods, which attempt to exhaust per-stream state on the server by opening many connections.

QUIC has built in some partial mitigations to its various vulnerabilities. For example, to minimize the impact of an amplification attack, the QUIC protocol sets a minimum size for the initial client hello message, thereby costing the attacker significant bandwidth to transmit many fake client hello messages. However, the server hello is still larger than the client hello, so the amplification asymmetry still exists.

Depending on the vulnerability, Gatekeeper may be able to complement the

mitigation techniques of the QUIC endpoint or provide entirely separate mitigating maneuvers. For example, to prevent client hellos from triggering amplification attacks, Gatekeeper can provide an upstream enforcement point for making sure minimum length requirements are met for client hellos, just like the QUIC server would, without wasting path or server resources. QUIC also prohibits the use of fragmentation at the network layer, which can also be checked by Gatekeeper. Such minimum length and protocol validation checks could be performed policy enforcement programs.

Additionally, to combat the protected network from being the *victim* of a QUIC amplification attack, i.e., the recipient of large server handshake packets, Gatekeeper can assign QUIC server hellos to a secondary bandwidth limit, bounding the effectiveness of the attack.

Other types of vulnerabilities, such as stream commitment attacks, require per-connection or per-stream state, and can only be mitigated with a significant portion of the QUIC protocol implemented. Due to memory restrictions as well as performance and maintainability considerations, this is not feasible in Gatekeeper.

3.5 ADVANCED POLICY TECHNIQUES

We now describe advanced techniques for writing policies. Although the architecture is in place for these more sophisticated tools to be realized, much of what is presented in this section serves as a jumping-off point for future research in the Gatekeeper ecosystem, including using a distributed database to analyze traffic at Grantor (Section 3.5.1), using port knocking for lightweight authorization (Section 3.5.2), and load balancing and path control (Section 3.5.3).

3.5.1 Flow Capture and Analysis

3.5.1.1 Design Overview

To this point, the role of Grantor servers has been to simply make policy decisions in response to requests and to decapsulate granted packets for their transmission to the destination server. However, Grantor servers have much more potential that has yet to be explored.

Due to their centralized location in the protected AS, Grantor servers acquire a global view of all ingress traffic into the protected AS. This data could be used to perform traffic analysis and inform policy alterations to mitigate attacks. Although there are already many tools available for traffic aggregation and analysis (e.g., Cisco NetFlow (2012), Packetbeat (2020)), most tools are based in dedicated routers and switches, require paid services, provide limited flexibility for new protocols and attacks, give few options for remedying issues when they are found, or some combination of all of these.

A traffic collection and analysis service that is built into Grantor servers would be beneficial in three ways. First, there are currently no standard tools for traf-

fic capture and analysis using DPDK network device drivers. Since Grantor uses DPDK, this compatibility is required. Second, Grantor is best placed to perform processing and sampling of traffic for the purposes of analysis. Since Grantor uses hardware and software optimizations to efficiently process traffic, layering other software on top to do further processing or sampling would likely be inefficient. Third, ultimately the results of the traffic analysis would ideally be used as feedback for creating or altering network policies, which ultimately must come back to Grantor anyway. Therefore, we decided that Grantor should sample traffic and directly access a database that it can use in policy contexts. We chose the Redis (Redis, 2020) database due to its efficient, in-memory implementation, its distributed design to work across multiple Grantor servers, and the fact that it is open source.

3.5.1.2 Case Study: Source Address Spoofing Detection

One potential application of a Grantor distributed database is detecting a form of source address spoofing, one of the major architectural issues that enable DDoS attacks (Section 1.3). Attackers that use a large botnet to launch attacks may choose to spoof source addresses in order to confuse per-flow rate control mechanisms, such as those in Gatekeeper. It would be beneficial to identify source addresses that are spoofed, either at policy decision or policy enforcement time, so that flows can be declined.

Because Gatekeeper uses a distributed overlay of vantage points, it has additional path and geographic information that it can use to try to detect spoofing. When Grantor receives an encapsulated packet from Gatekeeper, the source IP address of the outer header is the IP address of the back interface of Gatekeeper. This IP address uniquely identifies the vantage point in the Gatekeeper overlay. There-

fore, Grantor can associate the source IP address of the *inner* header, the client's IP address, with the vantage point through which it came. Figure 3.1 gives an overview of the association of flows to vantage points.

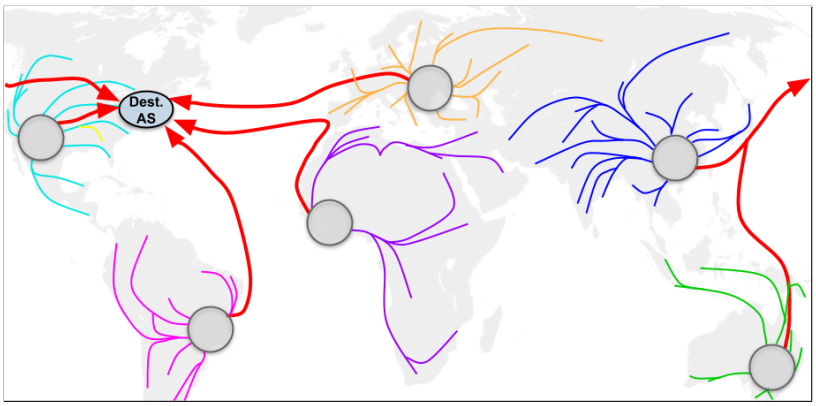


Figure 3.1: Example geographic overview of flows associated to VPs.

Given the static nature of deployments, the mapping of source network to its anycast-closest VP will be fairly static as well. This property allows Grantor to detect anomalous behavior whenever it receives packets from a client's source IP address that do not match the VP that was seen in a previous sample. An example of such a policy at work can be seen in Listing 3.4.

A caveat to this technique is that it gives attackers a potential handle for denying service to other flows. An attacker with *a priori* knowledge of the source address and assigned VP of a legitimate client could preemptively spoof that source address, so that requests from the legitimate client are declined by Grantor.

Because the test is not guaranteed to be accurate and could result in collateral damage to legitimate clients, this technique could be used as supplemental information to decide which rate to allot a flow, or could be used to de-prioritize flows that appear to have a spoofed source IP when Gatekeeper is overwhelmed by an attack. We leave a more thorough examination of the advantages and potential pit-

falls of such a source address spoofing detection, as well as other techniques that leverage geographical VP information, to future work.

```
function vp_changed(pkt_info)

    -- Get GK server's IP address.
    local outer_hdr = (struct ipv4_hdr *)pkt_info.outer_hdr
    local gk_src = tostring(outer_hdr.src_addr)

    -- Get client's IP address.
    local inner_hdr = (struct ipv4_hdr *)pkt_info.inner_hdr
    local cli_src = tostring(inner_hdr.src_addr)

    prev_gk_src = redis_db.get(cli_src)
    if prev_gk_src == nil then
        -- Client IP has not been seen before; add mapping.
        redis_db.set(cli_src, gk_src)
        return false
    end

    -- Return whether the Gatekeeper address is consistent.
    return prev_gk_src ~= gk_src
end

function lookup_policy(pkt_info, policy)

    -- Check whether the source address might be spoofed.
    if vp_changed(pkt_info) then
        -- Decline for 60s.
        return declined(policy, 60)
    end

    return default_granted(policy)
```

Listing 3.4: A policy decision program that detects changes in the mapping of client IP to VP, and declines flows whose VP changes.

3.5.1.3 Traffic Analysis with Machine Learning

Once the ability to sample, store, and analyze traffic data is established, statistical techniques and machine learning can be applied to detect network anomalies, which can be used to inform Gatekeeper policies.

Although the threat model for Gatekeeper is mostly concerned with infrastructure layer attacks, there do exist classes of attacks that use low-rate, benign-looking traffic to target vulnerabilities in protocol or application implementations. There is a significant body of work in network anomaly detection (Garcia-Teodoro et al., 2009) and intrusion detection via machine learning (Buczak & Guven, 2015) to pick up on such *low and slow* attacks. Given the ability of Gatekeeper to craft policies that inspect protocol header data, principal component analysis techniques can be used to mine network anomalies and report which traffic features are responsible for the anomaly (Lakhina et al., 2005). The resulting analysis could be used in an offline process to construct Gatekeeper policies, or perhaps in an online process to make adjustments to rate limits, for example.

In general, Grantor can be used as an insertion point for network analysis tools, such as attack detection research or industry tools for traffic analysis.

3.5.2 Port Knocking

The ideal scenario for a network capability system is for clients to be authorized for transmission *a priori*, before packets are put on the wire, so that the capability system can simply authenticate the client and verify its capability. Clearly, this sort of planning is not always feasible, since out-of-band capability distribution mechanisms are not convenient or easy to implement, not every interaction on the Internet is planned in advance, and it may not even be prescribed in the interaction model of the service to require pre-authorization of users.

Still, there is clearly a time and a place for such mechanisms. TLS client certificates, for example, allow servers to cryptographically authenticate clients based on a distributed digital certificate. Although the possession of a TLS client certifi-

cate does not imply that the client should be able to send at an arbitrary rate, it represents a form of authorization and can be the first step in a defense-in-depth strategy with multiple layers of protection.

For performance reasons, Gatekeeper servers are not currently equipped to perform TLS client authentication, although there is no technical reason why they could not do so. However, more lightweight forms of authorization, such as *port knocking* (Degraaf et al., 2005), are compatible with Gatekeeper. Port knocking traditionally refers to a technique that allows clients to open ports on a firewall by generating a specific sequence of connection attempts against particular ports. By default, ports protected by the port knocking mechanism appear closed. However, if a client generates a correct "knock," i.e., sequence of ports, then the firewall is opened for the client, allowing it to access services at valid ports.

In Gatekeeper's case, the Gatekeeper server itself represents the firewall. For each flow, a BPF program can be parameterized with a knock sequence, and state can be kept regarding the client's knocking progress using a simple counter. If the client completes the knock sequence, then their traffic is assigned the primary bandwidth of the link. Otherwise, the client's traffic is assigned the secondary bandwidth or dropped. An example is shown in Listing 3.5.

```
port_knocking(struct gk_bpf_pkt_ctx *ctx)
{
  struct state *state = (struct state *)pkt_ctx_to_cookie(ctx);
  struct packet *pkt = ctx_to_pkt(ctx);

  if (primary_limit(ctx, pkt) == DROP_PKT)
    return DROP_PKT; /* Primary budget exceeded. */

  if (state->correct_knocks == NUM_PORT_KNOCKS)
    goto forward;    /* Client already successfully port knocked. */

  if (ctx->l4_proto != IPPROTO_TCP)
    goto secondary;  /* Can only knock using TCP. */

  if (pkt->l4_len < sizeof(*tcp_hdr))
    return DROP_PKT; /* Malformed TCP header. */

  struct tcphdr *tcp_hdr = (struct tcphdr *)(pkt + offset)
  uint16_t knocked_port = tcp_hdr->d_port;

  /* Check whether this knock matches the next in the sequence. */
  if (knocked_port == state->ports[state->correct_knocks])
    state->correct_knocks++;
  else
    state->correct_knocks = 0;

  if (state->correct_knocks == NUM_PORT_KNOCKS)
    goto forward; /* This was the last knock needed. */

secondary:
  if (granted_pkt_test_2nd_limit(pkt->pkt_len) == false)
      return DROP_PKT;
forward:
  return FWD_PKT;
}
```

Listing 3.5: A policy enforcement program that uses port knocking to decide whether a flow should be assigned a primary or secondary bandwidth.

3.5.3 Load Balancing and Path Control

Policy enforcement programs can also be used to load balance flows to destination servers, either using encapsulation or direct delivery, instead of sending granted traffic through Grantor. This increases the robustness and performance of the protected network's services, while removing the need for dedicated load balancers. Since the load balancing is performed from VPs, Gatekeeper there-

fore becomes a scalable, geographically-distributed, DDoS-protected, centrally-administrated load balancer.

With the ability to perform encapsulation to an arbitrary destination in the protected network, Gatekeeper could also redirect flows. Flow redirection opens Gatekeeper up to new possibilities, such as integrating with intrusion detection systems or utilizing path control, which could be used to forward flows around congested links [or attacked links (Section 4.4)]. Combined with the fact that policies control (1) which flows are redirected, (2) the conditions under which they are redirected, and (3) the redirection destination, this technique could also be seen as a form of on-demand tunneling.

Overall, the richness of Gatekeeper policy decision and enforcement programs provides many opportunities for expressive control over the network capability system. We expect that the techniques previewed in this section will mature into tools that network operators can use in their deployments.

CHAPTER 4
Defending Against Next-Generation Attacks

4.1 OVERVIEW

We now turn to the future, and consider where malicious actors might take the DDoS field of play next. For the past two decades, attackers have been steadily increasing the effectiveness of their attacks along three axes: the magnitude of attacks, the frequency of attacks, and the sophistication of attacks (Arbor Networks, 2014; NETSCOUT, 2019; Neustar, 2020; AWS, 2020b). The trends are clear, and show no signs of slowing down. So how have attackers achieved this?

We can look to the state-of-the-art DDoS attack techniques for an answer, and perhaps no attack has greater exemplified the trends than *Mirai* (Antonakakis et al., 2017). The Mirai attack co-opts various IoT devices around the Internet, including cameras, DVRs, and routers, and uses them to infect more devices and launch a variety of attacks. Many of these devices are computation- and bandwidth-limited, and may reside in regions with low bandwidth capacity (Antonakakis et al., 2017). Despite these limitations, Mirai attacks have reached peak capacities of over 1.2 Tbps, and have targeted (and successfully brought down) entities that provide global DNS and cloud services, affecting large swaths of the Internet.

Still, Mirai-like attacks have not reached their full potential. This is because Mirai typically uses well-known TCP exhaustion and application layer attacks, which can be detected and mitigated using signature-based mechanisms. In one analysis (Antonakakis et al., 2017), 70% of the attack traffic was from non-infrastructure layer attacks. But not all attacks are like this. Consider the Crossfire attack (Kang et al., 2013). First described in 2013, Crossfire combines many of the properties of

Mirai with *stealth*. Crossfire uses low-rate, legitimate-looking traffic to launch a clandestine attack upon its victim, upping the ante in terms of attack sophistication.

Looking ahead, the network landscape is shifting in a direction that favors attacks like Mirai and Crossfire. New developments in the realms of user devices, access networks, and protocols are likely to further advantage attackers, and defensive solutions will desperately be needed.

In this chapter, we consider why Gatekeeper can be such a solution. In Section 4.2, we provide a primer on the Crossfire attack, which we have chosen to study as an example for two reasons. First, it it is especially relevant to this thesis because it enjoys architectural advantages over defensive systems. Second, the outlook for the growth of Crossfire attacks in the foreseeable future should worry network operators. We discuss this outlook in Section 4.3, where we describe the possibility of a perfect storm of factors – the proliferation of IoT devices, 5G networks, and IPv6 infrastructure – that could lead to highly disruptive Crossfire attacks. We then propose that Gatekeeper, with its architecturally-minded approach, is capable of mitigating or severely limiting the effectiveness of Crossfire attacks, shown in Section 4.4.

4.2 CROSSFIRE PRIMER

4.2.1 Attack Summary

Crossfire is a form of *link flooding attack* (LFA), i.e., an attack which seeks to deny service to legitimate clients by overwhelming one or more links in the network, as opposed to overwhelming only the network connection, CPU, or memory capacity of a target server directly. Crossfire was first described by Kang et al. (2013), which has led to a small but significant body of work about LFAs in general, including Kang et al. (2016); Lee et al. (2013); Smith & Schuchard (2018); Tran et al. (2019).

Although the locus of the attack is a set of one or more links, the victim of the attack is a *target area*, a geographic region which could be as small as a network enclave within an organization and as large as a country. As examples, target areas could be an individual university or the entire East coast of the United States.

To cut off service to a target area, the adversary seeks to find a set of *target links* that are geographically close to the target area. The adversary carefully chooses target links based on measurements that determine which of the relevant links likely carry a large share of the traffic to the target area. Therefore, launching a DDoS attack against this set of target links cuts off a target area from legitimate traffic.

An overview of the Crossfire attack is shown in Figure 4.1, and the relevant components for attack construction are summarized in Table 4.1. In order to calculate the set of target links, the adversary finds a set of public servers in the target area as well as a set of *decoy servers* in the vicinity of the target area. We call this set of public and decoy servers D. The adversary then selects a set of bots, B, and

sends multiple `traceroute` probes from every $b \in B$ to every $d \in D$. The result of these probes is a set of paths P, each of which is composed of links drawn from all encountered links, L. The adversary then extracts the *persistent* links $L' \subset L$ from these paths, i.e., those that always occur in the `traceroute` output across multiple trials for each pair (b, d). The set L' composes a network-layer *link map* (grey links in Figure 4.1), which represents the set of candidate links from which the target links will be chosen.

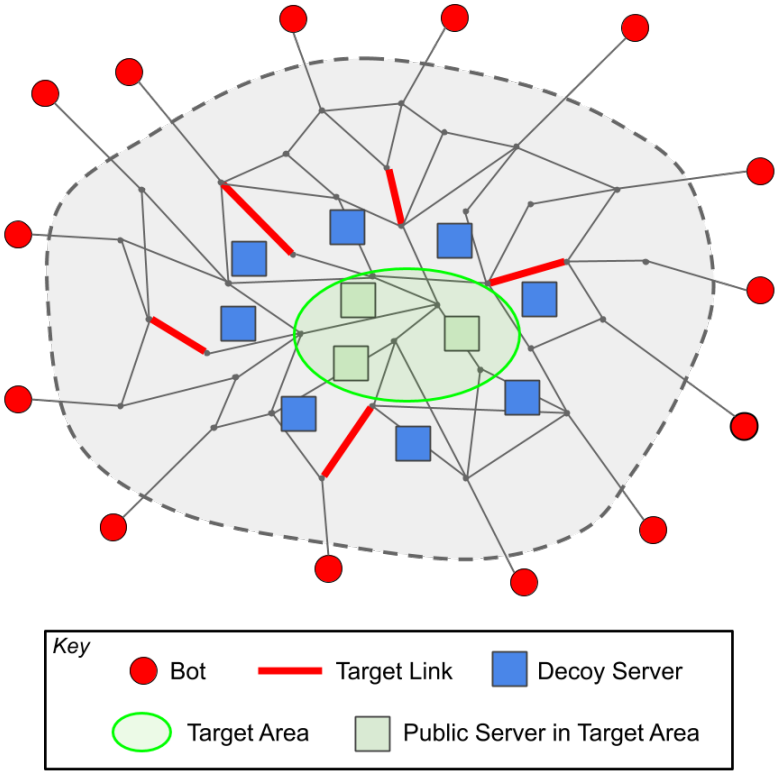


Figure 4.1: An overview of the Crossfire attack, illustrating the bots, target area, decoy servers, and target links. Based on diagram originally from Kang et al. (2013).

To select the target links, the adversary analyzes the map to find the links that would be most useful to flood with an attack. The metric used by the adversary is the *flow density* for each link $l \in L'$, which is defined as the number of paths $p \in P$

that cross l. Links with a high flow density are likely to carry the highest amount of legitimate traffic, and are therefore are the highest value links for an attacker to flood.

After selecting the set of target links $T \subset L'$, the adversary gives instructions to each bot, consisting of the set of decoy servers the bot should send to, as well as the rate at which it should send to each server. Bots use low-rate, legitimate-looking traffic that is destined for real, public servers, so that no individual flow looks out of the ordinary, but in aggregate the bots are coordinated in such a way that the target links are flooded. To circumvent defensive strategies and maintain attack persistence, the adversary can periodically rotate the set of target links that the bots flood, and/or rotate the set of bots performing the flooding.

Symbol	Definition
B	The set of bots
D	The set of public and decoy servers
P	The set of paths output by `traceroute` for all $(b \in B, d \in D)$ pairs
L	The set of links that compose paths $p \in P$
L'	The set of persistent links; subset of L; candidates for target links
T	The set of target links; subset of L'

Table 4.1: Summary of the components of the Crossfire attack construction.

The result is an attack that is exceptionally difficult to detect. Since link attacks like Crossfire cut off traffic at intermediate links in the client-server paths, the attack traffic does not reach the server itself, and therefore signature-based detection mechanisms that analyze traffic are not useful. Middleware appliances such as intrusion detection systems (IDS) that are placed further up the path may process some of the attack traffic, but only if some of the target links are chosen to be between the IDS and the public servers. Even then, other target links

may not be in that range. Furthermore, since the attack traffic uses low-rate flows with legitimate-looking traffic to legitimate destinations, the IDS is unlikely to flag traffic from Crossfire bots as anomalous. Automatic link failure detection mechanisms, such as those built into OSPF and BGP, are too slow to react to target link set changes that the adversary can perform.

Even if a victim could recognize that an attack was occurring, there is no mechanism for networks to protect themselves if the target links are outside of the administrative control of the network operator. A principal conclusion of Kang et al. (2013) is that coordination between networks is necessary to mitigate an attack if target links reside in different domains. These properties show that Crossfire takes advantage of the architectural issues that enable DDoS attacks, as we describe next.

4.2.2 Architectural Advantages of Crossfire

Crossfire is an interesting case study, as it relates to the architectural aspects of DDoS, as it enjoys several architectural advantages over defensive systems and exacerbates the architectural issues described in Section 1.3.

Inability to mitigate at attack locus (relates to Architectural Issue 1). Since the links used to cut off a target area can be outside of the target area's network, mitigation of the attack has to happen upstream, by another provider. Allowing a victim network to request mitigation from another network would require crossing management and trust boundaries, which is unlikely to happen in the near future given the decentralized operation of the Internet.

Seemingly benign nature of attack traffic (relates to Architectural Issue 2). Typically, attack traffic is unwanted, which provides at least a starting point for attack detection and fingerprinting mechanisms to distinguish it from traffic that is

legitimate. However, the nature of the attack traffic in Crossfire is more nuanced: it is unwanted in the sense that it is not deriving any real benefit for the client or server, and is dealing damage to the victim network. However, it is "wanted" in the sense that Crossfire uses seemingly legitimate traffic – valid uses of applications and protocols at appropriate rates – between real clients and servers, and therefore is indistinguishable from actual legitimate traffic. In this way, Crossfire exacerbates Architectural Issue 2 (*there is no mechanism to stop unwanted traffic*), since stopping unwanted traffic is predicated on being able to discern which traffic is unwanted.

No source spoofing required (relates to Architectural Issue 3). Since Crossfire uses legitimate-looking traffic, such as Web requests, it does not need to spoof source IP addresses. However, since the victim network does not necessarily have visibility into the attack traffic, the lack of source spoofing is not an additional advantage for the attacker beyond the appearance of legitimate traffic.

Widespread distribution of bots. (Relates to Architectural Issue 4). Similar to most DDoS attacks, Crossfire can be achieved using a distribution of bots from around the Internet. However, this is notably different from a closely related form of link attack, *Coremelt* (Studer & Perrig, 2009), where attack traffic is directly between bots and therefore the selection and utility of the attack locus is subject to the bot distribution.

Given these advantages, it is unsurprising that Crossfire attacks have made the leap from being theorized about in academic papers to hitting real-world infrastructure.

4.2.3 Recorded Crossfire Attacks

Malicious actors have already launched attacks similar to Crossfire in the Internet. In 2013, the anti-junk e-mail company Spamhaus was the victim of a massive (300 Gbps) DDoS attack. The locus of the attack was the links in the Internet exchange where Spamhaus traffic crossed on its way to Cloudflare, who provided DDoS protection services to Spamhaus (Prince, 2013; Bright, 2013). The attackers were able to find the IP addresses of peers in the IXP, and used those IP addresses to oversaturate some of the IXP links, leading to a successful attack. The Spamhaus attack used DNS amplification to obtain a high attack rate, so it did not quite use the legitimate-looking flows of traditional Crossfire, but it did focus its attack on upstream links, outside of where Spamhaus or Cloudflare could immediately see.

Two years later, in 2015, ProtonMail was the target of a week-long 50 Gbps attack (Patternson, 2015). The attackers set their sights on the links in the upstream ISPs that connected to the ProtonMail datacenters, and attacking them brought the datacenters offline. ProtonMail CEO Andy Yen described the event in stark terms: "We realized we were dealing with a different, far more scary attacker. One that didn't fit the pattern of any previous attack." It was only through the combined efforts of transit providers and other companies that ProtonMail was able to restore service.

These incidents demonstrate that large-scale link attacks are not only feasible, but can be very damaging in practice. However, there are not currently any comprehensive and deployable solutions to combat link attacks, as we describe next.

4.2.4 Previous Attempts at a Solution

Multiple partial solutions to link flooding attacks have been proposed. For example, CoDef (Lee et al., 2013) proposes collaborative rerouting, which permits victim networks to send "reroute" requests to source networks, asking them to use detour paths around congested routers or links. However, CoDef assumes the presence of a path identifier mechanism and inter-domain collaboration, both of which are not available in the current Internet architecture. SPIFFY (Kang et al., 2016) proposes to use mechanisms to increase the bandwidth of target links and observe the resulting traffic response, forcing attackers to either increase their attack cost or allow themselves to be differentiated from legitimate traffic. However, this solution assumes that the target links are within the jurisdiction of the deploying network. RADAR (Zheng et al., 2018) shows that an SDN-based approach can be used to detect and rate-limit link flooding attacks within a single network, but is not generalizable across networks.

For attack detection, LinkScope (Xue et al., 2014) proposes new end-to-end and hop-by-hop measurement techniques to detect LFAs, but does not provide a mitigation solution.

As a proactive defense, NetHide (Meier et al., 2018) proposes techniques for hiding the internal topology of a network to make the reconnaissance phase of link attacks difficult, but faces deployment hurdles in requiring the programmability of routers and does not provide a mitigation technique for target links that are able to be identified.

STRIDE (Hsiao et al., 2013) and SIBRA (Basescu et al., 2016) aim to use bandwidth isolation and reservation mechanisms to guarantee a minimum level of service for critical flows. While promising in their ability to combat DDoS attacks,

these proposals are not feasible in the current Internet architecture and would work best in novel network architectures such as SCION (Zhang et al., 2011).

Most recently, Nyx (Smith & Schuchard, 2018) proposes to use BGP poisoning to achieve isolation of traffic and route around congested links, obviating the need for trying to filter or throttle DDoS attacks. However, in response, Tran et al. (2019) suggested that the techniques used in Smith & Schuchard (2018) are not feasible in practice due to limitations in the implementations of BGP, and showed that adaptive adversaries can find new paths for its attacks.

The main problem with this body of work is that it consists of partial solutions and solutions that would require significant and expensive changes to the Internet architecture to deploy. Unfortunately, as we describe next, link flooding attacks may soon become more commonplace in the Internet, accelerating the need for a comprehensive and deployable solution.

4.3 THE PERFECT STORM

The ecosystem of the Internet's infrastructure and connected devices may soon be changing in a way that makes Crossfire attacks even more feasible for attackers to launch. There are at least three recent developments that could provide new attack opportunities.

The proliferation of IoT devices. Recent years have seen the development of cheap and simple ways of constructing botnets composed of IoT devices. Although IoT botnets existed before 2016, that year saw the introduction of the Mirai botnet, which (along with its derivatives) has become the foremost example. The original Mirai botnet reached a population of 200,000-300,000 bots, but it is estimated that at least around 20 million such devices exist in the Internet as of 2020 (Guo & Heidemann, 2020). The number of devices will continue to grow, and the diversity of these devices will continue to grow, which makes finding and patching security vulnerabilities like those exploited by Mirai quite difficult.

Such a wealth of potential bots is especially useful for Crossfire, since IoT devices are distributed throughout the world, providing a diverse snapshot of paths that lead to a target area. Further, the modest network capacity of IoT devices is well-suited for Crossfire, since it can achieve successful attacks using just a few Kbps (Kang et al., 2013).

The introduction of 5G networks. The rollout of the fifth generation of mobile networks has begun across the world (Xu et al., 2020). 5G networks will provide higher bit-rates, lower latencies, and greater reliability, enabling many real-time and virtual applications. The 5G was built in part to support features for IoT, such as systems for slicing virtualization and infrastructure and access technologies for the massive density of traffic between IoT devices (3GPP Org., 2019).

Although Crossfire may not fully utilize the bandwidth increases enabled by 5G networks (since it uses low-rate flows to its advantage), the new network support for billions of new devices provides Crossfire with the infrastructure needed to leverage the billions of new potential bots at its disposal.

The deployment of IPv6. Although support for IPv6 began to roll out in the late 1990s, adoption did not significantly pick up steam until around 2009 (Czyz et al., 2014a), and there has been a steady increase in adoption since then. By the numbers: 30% of Google's users access their services using IPv6 as of 2020, and has been increasingly approximately 5% per year since 2014 (Google, 2020a). IPv6 usage in IXPs is on an upward trend as well: IX.br has measured peak IPv6 throughput at over 500 Gbps (IX.br, 2020), and 5-10% of the average traffic passing through SeattleIX is IPv6, with a peak around 16% (SeattleIX, 2020). In terms of adoption by network, mobile carriers and telecoms are generally leading the way, with India-based Reliance Jio at 90% IPv6 deployment among its users, and all combined US carriers at 86% (World IPv6 Launch, 2020).

IPv6 brings many technical strengths to the Internet landscape, but burgeoning deployments also present weaknesses. Rapidly rising deployment rates in both content networks and eyeball networks across the globe, especially mobile providers, are driving up rates of IPv6 traffic, but management tools, hardware, and native deployments ("IPv6 only") of links and services may not be adequately provisioned. Network configuration tools as well as DDoS detection and mitigation tools have had more time, resources, and events to be refined, and IPv6 will have its own unique challenges. For example, state exhaustion attacks may become more prevalent due to the expanded address space of IPv6.

Anecdotally, in our implementation of Gatekeeper, we found both hardware

and software support for filters for IPv6 traffic to be more limited. In our conversations with network operators, we found that networks are incrementally shifting parts of their services and infrastructure to IPv6 to test the waters before larger deployments are realized. Such partial migrations may leave parts of their infrastructure underprovisioned or otherwise not ready to face an IPv6-based attack.

In summary, the combination of the emergence of IoT devices, 5G networks, and IPv6 presents a scenario where the number of devices and their aggregate capacity is rapidly increasing, while the underlying network infrastructure is simultaneously undergoing a radical shift. Such a confluence of factors may lead to a perfect storm of conditions for large scale link attacks like Crossfire to become commonplace.

As the Internet hurdles toward this next generation, there are no deployable, comprehensive defensive strategies for combating Crossfire-like attacks. However, there may be hope: architectural approaches to mitigation, such as Gatekeeper, may provide the techniques needed to defend against large-scale link attacks.

4.4 CROSSFIRE DEFENSE WITH GATEKEEPER

Recall that a Crossfire attack consists of three phases: constructing the link map, coordinating bots to flood target links, and periodically rotating the set of target links to maintain attack persistence. Gatekeeper offers strategies for combating attackers at each of these phases.

Crucially, some of these strategies depend on the ability of Gatekeeper to be deployed in cloud environments. Clouds offer an advantage to Gatekeeper in two ways. First, due to the massive infrastructure investment of large cloud providers, e.g., Amazon, Microsoft, and Google, significant portions of cloud paths are independent from, and more reliable than, public Internet paths, to the extent that such paths have been proposed to improve the quality of service of many applications (Haq et al., 2017, 2020). Gatekeeper leverages the independent nature of these paths to circumvent links that are targeted by adversaries, or else require that Crossfire botnets are composed of cloud VMs. Second, due to the flexibility of deployment in the cloud, Gatekeeper VPs can be quickly bootstrapped, analyzed for their topological characteristics, and put into service during an attack. Because of these advantages, some of the strategies discussed next may or may not be applicable when Gatekeeper is deployed in other environments, such as IXPs.

To support our findings, we employed a real-world measurement study that emulates the attack setup of a Crossfire adversary, the details of which are presented in Section 4.4.1. Then, we show the advantages that Gatekeeper has over Crossfire attacks. First, Gatekeeper can disrupt the attacker's ability to construct a useful link map (Section 4.4.2). When an attacker floods links, many Gatekeeper paths may not cross any of the target links chosen from the link map, even when rotating sets of links are used (Section 4.4.3). Finally, even when an attack is suc-

cessful, Gatekeeper can use a moving target defense to reduce the likelihood of legitimate traffic crossing saturated links (Section 4.4.4).

Note that we do not quantitatively compare our results to the previous attempts at Crossfire solutions (Section 4.2), as the measurement study simply evaluates the feasibility of the proposed Gatekeeper defense. Additionally, since all of the previous attempts at solutions are either (1) partial solutions with much different assumptions (e.g., the ability to collaborate between networks), or (2) not feasible in the current architecture, a direct comparison is not practical.

4.4.1 Measurement Study Setup

In order to evaluate how Gatekeeper can respond to a a Crossfire attack, we needed to emulate the steps that an actual attacker would perform, as described in Section 4.2. Although for ethical and practical reasons we did not launch a large-scale link attack, we developed the Crossfire link map construction algorithm as described in Kang et al. (2013), which includes algorithms to extract a set of persistent links from `traceroute` results, find target links with high flow densities, and pick sets of target links to use in rotating attacks to maintain attack persistence.

For a target area, we selected a geographic region that approximately corresponds to the city of Boston, Massachusetts, USA. Notably, Boston is home to many higher education institutions, as well as medical, biotechnology, and general high technology and innovation industries, making it a well-connected region in terms of network infrastructure as well as technological importance. The results of the study may or may not apply to regions with poorer connectivity properties.

We also selected approximately 100 Looking Glass nodes using the CAIDA Periscope (Giotsas et al., 2016) as a proxy for the Crossfire botnet. These nodes are

distributed across the world, and are available for public use to run `traceroute` for Internet measurement purposes. We limited our botnet to the 100 nodes with the consistently highest link persistence on the path from each bot to the target area. However, to increase the geographic distribution of bots, we limited our set to at most one bot per city.

To compose our set of decoy servers, we found approximately 60 public servers in the networks of institutions around the target area. Although Kang et al. (2013) recommend finding such decoy servers through "port-scanning," we took a multi-step approach:

1. Perform *subdomain enumeration* to find publicly accessible subdomains from institutions around the target area. To avoid selecting decoy servers that are not physically near the target area, such as for subdomains that resolve to cloud services, cross-referencing should be performed with multiple tools to estimate the approximate geographic location of each IP.

2. For each IP address corresponding to a subdomain, perform a `traceroute` to that IP address in order to test whether probes are administratively blocked by the network.

3. If `traceroute` probes succeed, then look for other servers in the same network, since there may be other servers that are publicly accessible but do not have DNS entries. To find these servers, we use `nmap` (Lyon, 2009) against the `/24` network prefix of the IP address found as a guess of what other servers may exist with the same administrative policies.

4. Run `traceroute` to each of the servers found by `nmap` to verify that they can be probed.

The target area, bots, and decoy servers are sufficient for the Crossfire setup and attack. However, to evaluate the effect of Gatekeeper defensive strategies, we simulate a Gatekeeper deployment using cloud nodes in AWS to represent 6 geographically distributed vantage points, in Ohio (US), Frankfurt (EU-1), Paris (EU-2), Sydney (AU), Mumbai (IN), and São Paulo (BR). We refer to these VPs as the set V, We chose nodes in these regions to simulate a global Gatekeeper deployment, but also chose two VPs in Europe (Frankfurt and Paris) to be able to compare the value of VPs that are geographically close.

Note that in this evaluation, Gatekeeper is not actually deployed. We simply use representative vantage point locations to determine what Gatekeeper's effect would be on the topological assumptions and advantages of Crossfire, were Gatekeeper actually deployed there.

Figure 4.2: Distribution of Looking Glass nodes for bots (blue pins) and simulated Gatekeeper vantage points (yellow icons).

An overview of the distribution of bots and VPs is shown in Figure 4.2. With this setup, we then evaluated how Gatekeeper can defend against Crossfire attacks during the link map construction, link flooding, and attack rotation phases. We discuss Gatekeeper's defense to each of the phases next.

4.4.2 Link Map Disruption

In constructing Crossfire link maps, adversaries may use `traceroute` results that are from bots to servers protected by Gatekeeper. This can occur either when the Gatekeeper-protected AS is in the target area, or when the Gatekeeper-protected AS can be used for decoy servers *around* a target area. If the adversary knows that certain servers are protected by Gatekeeper, either because such information is public or through path analysis, then they might choose to specifically avoid those servers if there are a sufficient number of other decoy servers available. However, if these paths are included in the link map construction, then the utility of the result may be diminished in terms of candidate target links for the following reasons.

The effect of encapsulation. Since traffic between Gatekeeper and Grantor is encapsulated, the attacker has no visibility into that section of the path. This is due to the way that `traceroute` is typically implemented: using probes with incrementally increasing time-to-live (TTL) values, causing routers who see TTL values of 0 to return ICMP `Time Exceeded` messages back to the source. This sequence of messages allows senders to piece together the path to the destination. However, between Gatekeeper and Grantor, the IP header with this crafted TTL value is submerged in another IP header, so routers do not inspect it and therefore do not send the ICMP messages to the sender. Additionally, Gatekeeper and Grantor servers at the ends of the tunnel do not inspect this value, and Grantor is expected to be deployed on the same network segment as the ultimate destination. Therefore, after the last hop before Gatekeeper, the only hop along the path that will reply to the probe is the destination server (Figure 4.3). This obscures the adversary's view into the links around the target area. Since this is an area where potentially many paths converge, the pool of candidate target links may be reduced.

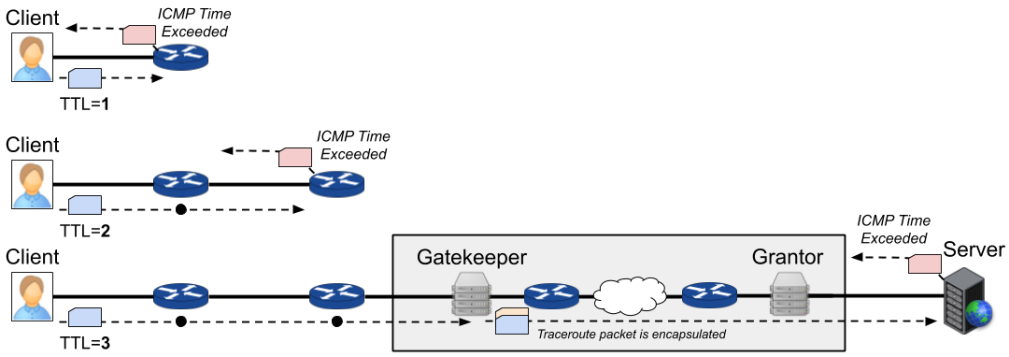


Figure 4.3: Handling of `traceroute` probes around the Gatekeeper to Grantor tunnel.

The effect of anycast. In a Gatekeeper deployment, the only links that are visible to the adversary's `traceroute` bots are the links leading up to the VP. These links have indeed been targeted by large-scale link attacks in the past (Prince, 2013; Bright, 2013). However, in that attack, the IP address of the router inside of the IXP was exposed, and therefore the adversary could use *all* of its bots to target the links leading up that router as a part of the flood. Since traffic destined for Gatekeeper-protected ASes are anycast to the topologically nearest VP, the adversary's pool of bots is divided, limiting the bots that could attack a given VP to those that are geographically close. We could not measure this effect in our study, since dividing our pool of bots among the VPs would create some sample sizes that were too small to be relevant, especially for VPs that had only a few bots (e.g., Australia). Even if the pool of bots were large enough to saturate a critical mass of links leading to a VP, that is only one entry point for Gatekeeper traffic. Traffic that uses other VPs in a global deployment would be unaffected.

The effect of policies. Some Gatekeeper deployments may choose to leverage the ability of Gatekeeper to deliver granted packets directly to destinations, instead of first through Grantor servers (Section 2.2.3). In this case, encapsulation

is not used along the Gatekeeper-Grantor channel. However, Gatekeeper policies can be formed to administratively prohibit `traceroute` along this path. Although there is no definitive signature of `traceroute` packets, which are typically either be UDP or ICMP datagrams, there are heuristics that can be used. For example, they typically have low TTL values in order to trigger responses from routers. A sample policy decision program in Lua that captures packets likely to be `traceroute` and declines the flow is shown in Listing 4.1. A similar technique could be used in policy enforcement programs for flows that have already been granted.

```
local function policy_lookup(pkt_info)

    if pkt_info.l4_proto == policylib.c.ICMP then

        -- Snip: extract IPv4 header and ICMP type/code.

        -- Disable traceroute through ICMP into network.
        if ipv4_hdr.time_to_live < 16 and
                icmp_type == ICMP_ECHO_REQUEST_TYPE and
                icmp_code == ICMP_ECHO_REQUEST_CODE then
            return declined
        end
    end

    return default
```

Listing 4.1: A policy decision program in Lua that declines flows that appear to use `traceroute`.

The defenses discussed so far only relate to adversaries that try to use servers protected by Gatekeeper as part of the link map construction. But this is not necessarily the case. Adversaries may attempt to victimize a Gatekeeper-protected network by building a link map using decoy servers that are known to be geographically close to the victim network. We discuss this scenario next.

4.4.3 Diversity of Cloud Paths From Gatekeeper

Crossfire does not rely solely on servers in the target area to perform the link map construction. If an attacker knew that Gatekeeper was protecting a target network, then it could construct the link map solely using decoy servers in unprotected networks, with the assumption that the target links would overlap with links in the paths from Gatekeeper VPs to the target area. However, even if an attacker takes this approach, the effectiveness of this link map is limited due to the topological differences of Gatekeeper paths, at least when Gatekeeper is deployed in a cloud.

To measure this effect, we analyzed the results of the study described in Section 4.4.1. We used the results to construct a link map ($|L'| = 843$), and picked a set of target links according to the attack construction described in Section 4.2. Figure 4.4 shows the persistent links ranked by flow density, and highlights the target links that were chosen, where $|T| = 20$.

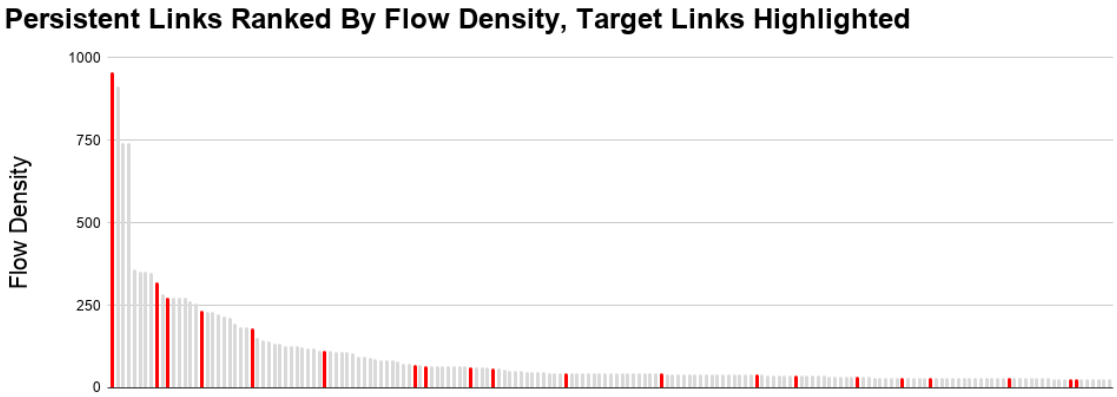


Figure 4.4: The target links T chosen from the set of persistent links found in the measurement study, where $|T| = 20$.

To investigate whether Gatekeeper would enable client traffic to circumvent these target links, we then ran `traceroute` probes from our set V of vantage points

in AWS to a sample of the target area servers. We then compared the $v \in V$ to $d \in D$ paths to check whether they crossed the target links constructed by the Crossfire attack. Our four main findings are:

1. The number of paths that cross target links when directed through a Gatekeeper VP is much fewer than the number of paths that cross target links when forwarded directly to the target area.

2. Even if Gatekeeper paths cross target links, they are not necessarily crossing the links with the highest flow density, so attackers would need to flood more links in order to maintain an effective attack, raising attack cost.

3. Findings (1) and (2) are true for both a static attack of 20 target links and a rolling attack of three sets of 10 target links.

4. Moving target-based defenses could be effective due to the marginal path diversity obtained from different VPs.

We now elaborate on these findings.

Cloud paths from Gatekeeper cross fewer target links. We use the *degradation ratio* (Kang et al., 2013) to measure the effectiveness of a set of target links in capturing traffic to the target area. For a given number of target links, the degradation ratio is the proportion of paths that cross target links (the set P_T) to the overall set of paths P: $\frac{|P_T|}{|P|}$. In other words, it is the fraction of the paths to a target area that can expect to run into at least one saturated, targeted link during an attack. A successful attack will create a high degradation ratio.

Figure 4.5 shows the degradation ratios from our measurement study for different sizes of T, from $1 \leq |T| \leq 50$. The **No VP** series represents the degradation

ratios for the traditional Crossfire attack, without any VPs. In general, only a small set T is needed to force most paths through a target link: 75% of paths to the target area cross a target link when $|T| = 10$. This result is consistent with Kang et al. (2013).

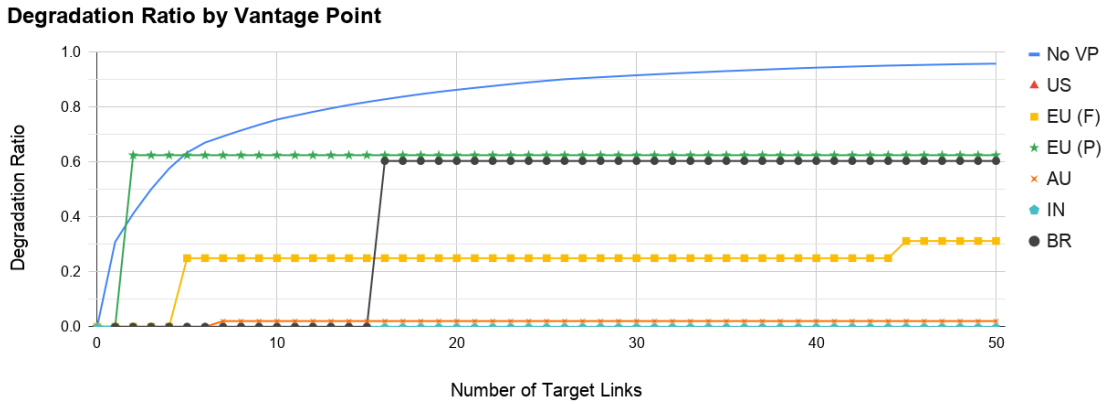


Figure 4.5: The degradation ratios when paths are forwarded directly to a target area and forwarded through VPs.

Figure 4.5 also shows the degradation ratios for all (v, d) paths, from each VP to the decoy servers in the target area. This result emulates a Gatekeeper defense, in which all client traffic is funneled through a vantage point before being sent to the target area. We found that the degradation ratios for paths from VPs are mostly less than what they would be if forwarded directly to the target area, in some cases considerably so. This means that client traffic forwarded through Gatekeeper in a cloud deployment would largely not cross target links.

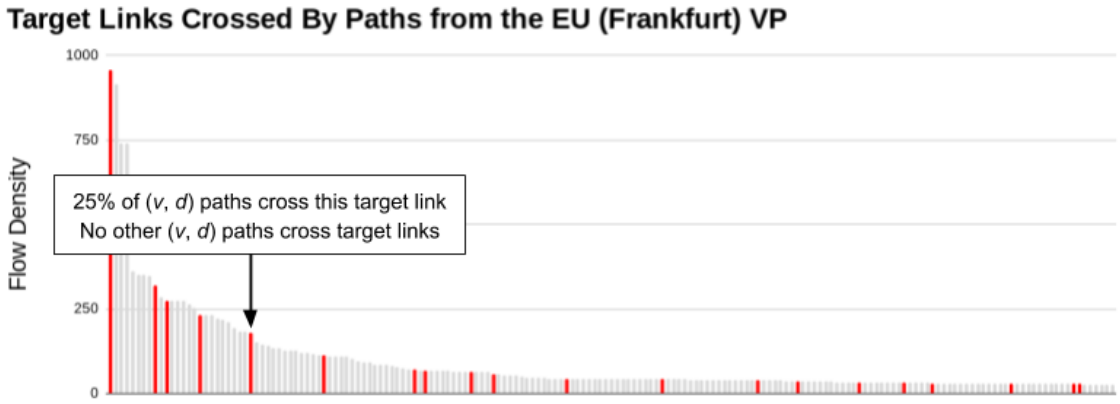
For example, no paths from the US (Ohio) or the IN (Mumbai) VPs crossed any target links. Only one path from AU (Sydney) crossed a target link. The EU (Paris) and BR (São Paulo) VPs saw the most overlap with target links: over 60% of paths in both cases. However, we must also consider *which* links are being crossed by such paths.

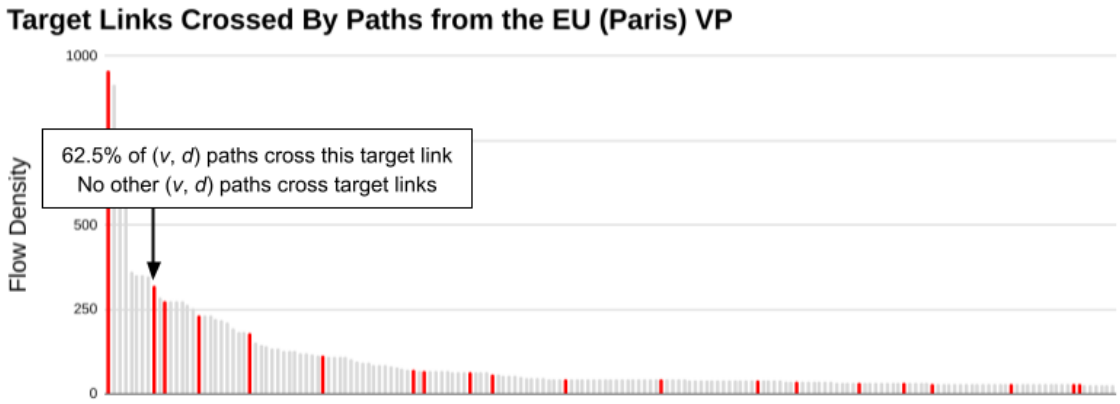
Target links in cloud paths from Gatekeeper are not necessarily high-value links. For an adversary, it is desirable to minimize the number of target links used to launch an attack. More target links can require more bots to make the attack successful, since Crossfire attacks use low-intensity flows to maintain attack persistence. However, when Gatekeeper paths cross target links, they are not necessarily high value links in terms of flow density.

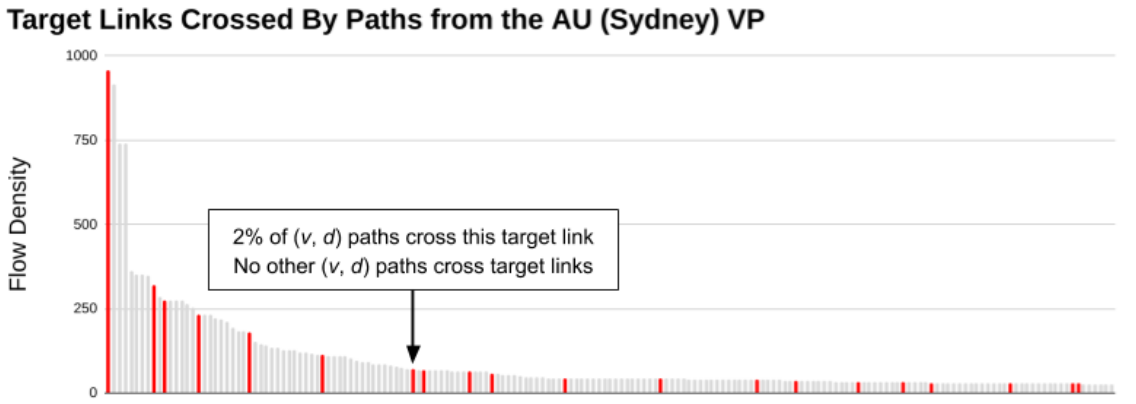
To visualize this, we can drill down into the case where the number of target links is 20. At that level of attack, we can view which target links the VP-to-target area paths are crossing in Figure 4.6. Each plot is from the perspective of a different VP, and shows all persistent links, ranked by flow density, with target links highlighted. The arrows point to target links crossed by paths from the VP.

Paths from EU (Frankfurt), EU (Paris), and AU (Sydney) cross only a single target link in each case, but all cross fairly high-value target links: the fifth, second, and seventh target links by flow density, respectively. We previously noted that 60% of the flows from the BR (São Paulo) VP crossed target links. However, all of those flows crossed a single, relatively low value target-link: the 16th. The flow density of this link is just 31, compared to 957, 319, and 275 of the top three persistent links by flow density. Overall, these results show that (1) even when paths from VPs in clouds cross target links, they do not cross many of the links in the set, and (2) the paths do not necessarily cross high-value target links, forcing the adversary to attack more links, raising attack cost.

We omit charts for the US (Ohio) and IN (Mumbai) VPs, since no paths crossed target links. However, we note that this result is ideal for a Crossfire defense.


Target Links Crossed By Paths from the EU (Frankfurt) VP
Flow Density
1000
750
250
0
25% of (v, d) paths cross this target link
No other (v, d) paths cross target links


Target Links Crossed By Paths from the EU (Paris) VP
Flow Density
1000
500
250
0
62.5% of (v, d) paths cross this target link
No other (v, d) paths cross target links


Target Links Crossed By Paths from the AU (Sydney) VP
Flow Density
1000
750
500
250
0
2% of (v, d) paths cross this target link
No other (v, d) paths cross target links

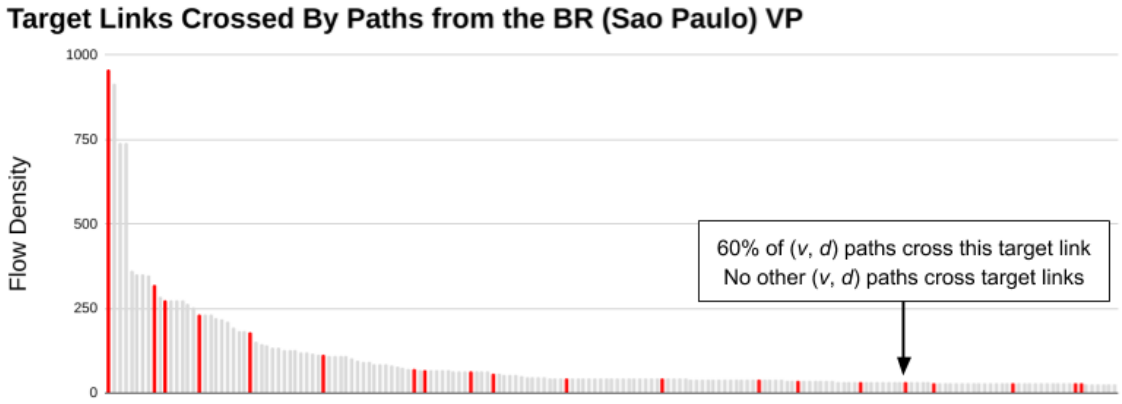
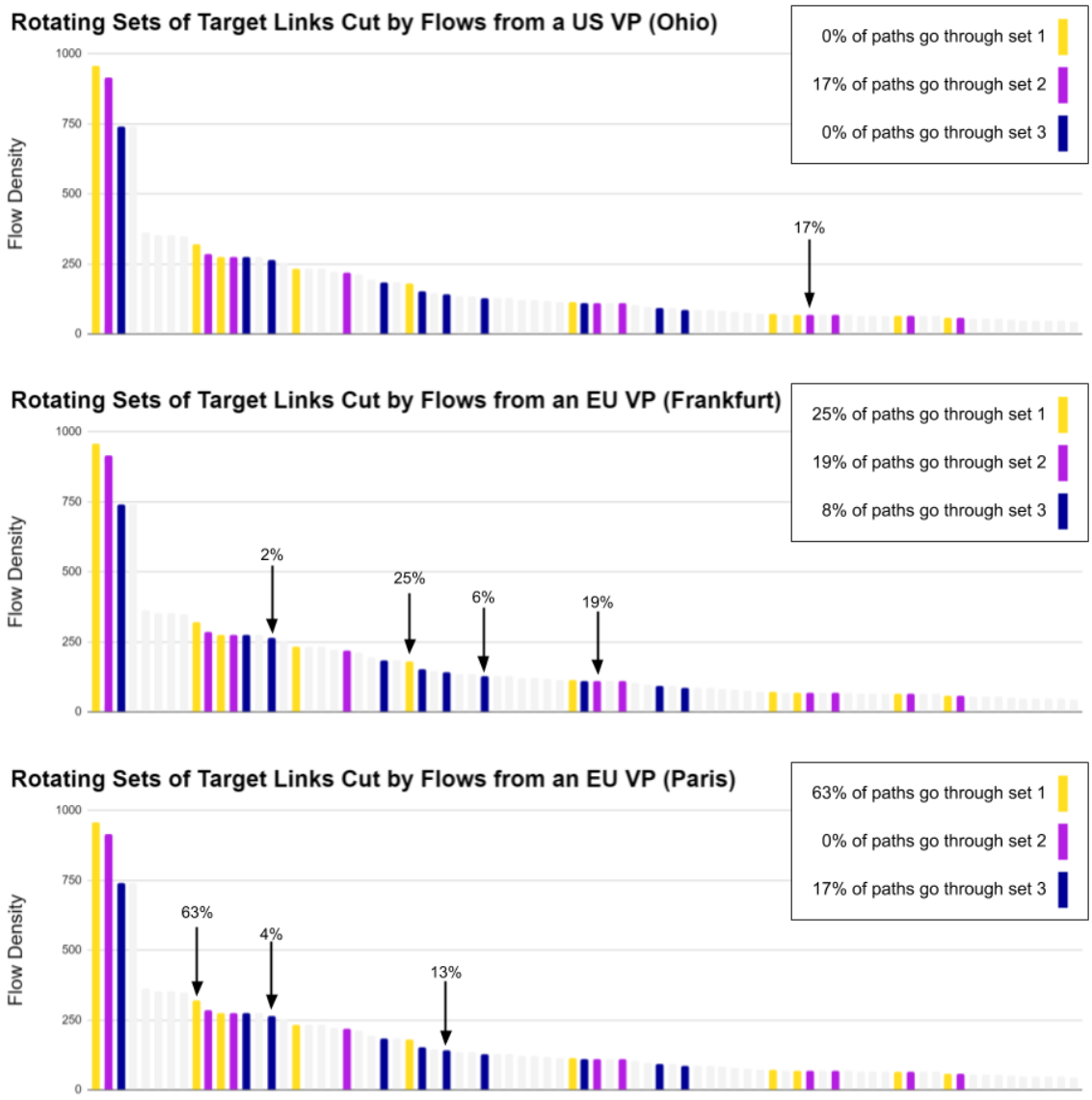


Figure 4.6: Target links cut by flows from the various VPs, where $|T| = 20$. Paths from VPs do not cross many target links, and may not cross high-value target links.

Cloud paths from Gatekeeper can circumvent rotating sets of target links. To maintain attack persistence, adversaries can use disjoint sets of target links to avoid mitigations. Each target link set is chosen independently, after removing all links chosen in previous sets. We evaluated Gatekeeper paths in this context, and found that such rolling attack strategies do not provoke higher degradation ratios.

Figure 4.7 again shows the persistent links ranked by flow density, but this time highlights three sets of 10 target links. For each VP, we note the percentage of paths that go through each set of target links. Note that paths can go through more than one target link, which is why the path percentage may not equal the sum of the fractions of the links. In general, the findings are similar to the 20 target link case. Attackers can capture a good deal of Gatekeeper paths in some cases [63% for one set from EU (Paris) and 50% for one set from BR (São Paulo)]. In most cases, however, zero or few paths contain target links, and overall just 11 out of the 30 target links capture any paths at all. Therefore, rolling attacks do not give the attacker a significant advantage in terms of degradation ratio.


Rotating Sets of Target Links Cut by Flows from a US VP (Ohio)
0% of paths go through set 1
17% of paths go through set 2
0% of paths go through set 3
Flow Density
1000
750
500
250
0
17%
Rotating Sets of Target Links Cut by Flows from an EU VP (Frankfurt)
25% of paths go through set 1
19% of paths go through set 2
8% of paths go through set 3
Flow Density
2%
25%
6%
19%
Rotating Sets of Target Links Cut by Flows from an EU VP (Paris)
63% of paths go through set 1
0% of paths go through set 2
17% of paths go through set 3
Flow Density
63%
4%
13%

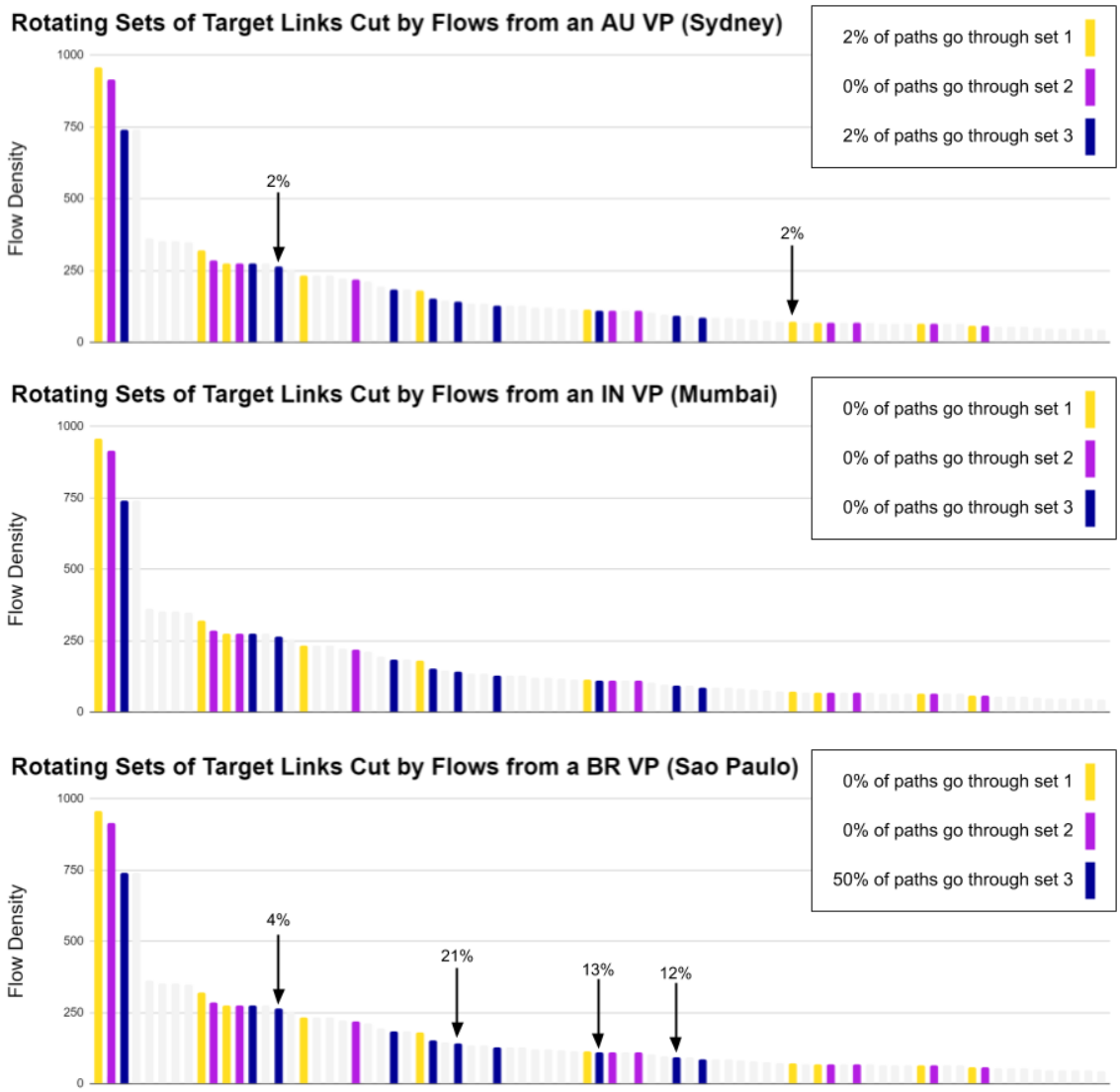


Figure 4.7: Target links cut by flows from the various VPs, where there are three sets of 10 target links. The arrows point to target links crossed by paths from the VP, and the label indicates the fraction of paths that cross that link.

Clearly, cloud paths are independent enough from Internet paths that attackers must flood more links to increase the likelihood of a successful attack, and even if they do so, there are no guarantees. However, even for the attacks that are successful, Gatekeeper can respond by utilizing a *moving target defense*.

4.4.4 Moving Target Defense

A major advantage for Gatekeeper is the ability to leverage the agility and flexibility of the cloud to deploy new vantage points. While the planning and management of deployments at IXPs is more of a long-term process, often consisting of establishing a physical presence[1] and contractual agreements, cloud deployments can be created quickly. Therefore, we ask: is it feasible to deploy new VPs during an attack as a mitigating maneuver?

Such a maneuver would be a moving target defense, similar in spirit to several previous proposals (Geng & Whinston, 2000; Wang et al., 2014; Venkatesan et al., 2016) which range in complexity from simply changing the victim's IP address to shuffling clients that are affected by attacks to intermediate, cloud-based proxies. Unlike these proposals, Gatekeeper does not use a lookup or authentication server to map clients to intermediate proxies, but rather utilizes the anycast enabled by BGP route prefix announcements to map clients to their nearest VP.

The "moving target" in such a scenario is the set of paths that carry traffic from Gatekeeper VPs to the destination network. The adversary's goal is to pick a set of target links that cover as many of these paths as possible. Therefore, if Gatekeeper had the ability to alter this set of paths, the adversary would be forced to change their strategy to find a new set of effective links. The effectiveness of the moving target defense will depend on the ability of Gatekeeper to (1) to maximize the independence of these paths from those that already exist (and are being saturated) and (2) alter the set of paths significantly more quickly than the adversary can find them and attack them.

To evaluate the independence of paths from new VPs, we did a pairwise com-

[1] Except in the case of remote peering (Chatzis et al., 2013).

parison of the links on paths from each VP to the destination network. In general, we found that there is a strong possibility that the cloud paths from new VPs are quite disjoint from existing paths. The size of the intersection between each VP's set of links is shown in the form of a heat map in Figure 4.8. The diagonal of the heat map represents the number of persistent links in paths from each VP to the destination network. We find that in our measurement environment, the size of the intersection of the links between each VP is fairly small. For example, there were only 11 persistent links in common between (1) all paths from the US VP to the target area and (2) all paths from a European VP to the target area. This indicates that the paths from each VP contain largely disjoint sets of links. This is perhaps an intuitive result for VPs that are geographically far apart, so we also compared two VPs that are relatively close: EU (Frankfurt) and EU (Paris). We found that the size of the link set intersection between these two VPs is very small (37 links) and not significantly different from all other pairs of VPs. This means that it can be valuable to create new VPs as part of a moving target defense even when they are geographically close to existing VPs.

	US	EU-1	EU-2	AU	IN	BR
US	522	12	11	13	13	12
EU-1	12	261	37	17	16	30
EU-2	11	37	238	15	14	29
AU	13	17	15	394	23	16
IN	13	16	14	23	322	15
BR	12	30	29	16	15	232

Figure 4.8: A heat map showing the size of the intersection between sets of persistent links on paths from different VPs.

The second key concern, regarding the agility of the moving target defense, depends on the speed and accuracy of actions from both the defensive system and the attacker. The defensive system must be able to quickly bootstrap a new vantage point, but also must be able to propagate BGP prefix announcements sufficiently quickly to forward traffic to the new VP. In most cases, the convergence time of BGP update messages is below 5 seconds, although it can take on the order of minutes in the tail (Tran et al., 2019; Cox, 2019). On the other hand, the adversary must quickly and accurately find a new set of target links. In Tran et al. (2019), the authors show that a *detour-learning* attack can quickly and accurately find new target links after routing changes affect the topology of the attack surface, but the technique presented depends on the ability of the adversary to use `traceroute`. As shown in Section 4.4.2, Gatekeeper disrupts `traceroute` and hinders the construction of the link map due to the tunneling of data plane packets throughout much of the path to the destination server. Therefore, it is unclear how valuable a moving target defense using Gatekeeper would be; we leave a more thorough analysis of defensive techniques and adversary reactions to future work.

In general, Gatekeeper provides several tools and techniques to combat large-scale link attacks like Crossfire. In the end, the most promising solution may be to leverage the unique path properties of cloud deployments with the architectural benefits of Gatekeeper to neutralize the rising threat of large-scale link attacks in the near future.

CHAPTER 5
Conclusions

5.1 DEPLOYMENTS

A recurring theme in the design and implementation of Gatekeeper is deployability. At publication time, Gatekeeper is undergoing the initial stages of deployment by two networks: Digirati and Mail.Ru.

Digirati. A Brazilian Internet company that provides Website hosting and domain registration services, Digirati was the initial source of funding for Gatekeeper and was interested in deploying it for 10 Gbps of protection. The Digirati deployment drove the development of policies, forming the basis of the network profile-to-program heuristic described in Section 3.3. It also motivated the development of Drib (Nathan, 2020), a Rust tool that lets operators manage IPv4 and IPv6 address for use in configuring Gatekeeper.

Mail.Ru. The second ongoing deployment is by Mail.Ru, a major Internet company in Russia that owns and manages three large social media networks, and provides email, search, and e-commerce services. After searching for available open source solutions for DDoS mitigation, Mail.Ru settled on Gatekeeper as a component in their overall defense architecture, and is currently working on a 1 Tbps deployment.

5.2 CLOSING REMARKS

From time to time, issues arise that challenge the Internet architecture. The Internet is constantly growing in scale and scope, including the emergence of new user bases [e.g., Africa (Tuerk, 2020)], devices (e.g., IoT), infrastructure and hardware

capabilities [e.g., 5G, SmartNICs (Firestone et al., 2018)], and applications and services [e.g., telesurgery (Gupta et al., 2019)].

These developments push, pull, and poke the architecture in new ways, and force us to consider how, for example, we can better *program* networks [e.g., ANTS (Wetherall et al., 1998)], *manage* networks [e.g., SDN (Casado et al., 2007)], and *evolve* networks [e.g., XIA (Han et al., 2012)]. Along the same lines, the increasing financial, political, and social importance of the Internet has driven malicious actors to seize on DDoS attacks as a disruptive tool, and so the networking community has also been forced to reconcile the architecture with techniques to *protect* networks.

However, there is a major difficulty in creating solutions that tackle these high-level problems in the Internet: the architecture itself is famously resistant to change and intricately anchored to certain principles and protocols. For example, the distributed administration of the Internet makes it exceptionally difficult to reach consensus among the Internet's myriad stakeholders. Without consensus, change cannot happen. Additionally, the architecture's reliance on IP as the *de-facto* network layer protocol makes it prohibitively expensive to remove that dependency all the way down to the infrastructure level. The number of solutions that have broached these challenges and failed are too numerous to count. But to be fair, a select few have succeeded in achieving the escape velocity needed to see actual deployment (e.g., SDN).

Indeed, there is a tension between architectural "purity" and deployability (Anderson et al., 2005). Purists argue against the mechanisms that massage the architecture into being amenable to the deployment of new solutions, such as overlays, middleboxes, and virtual networks. And there is a convincing argument for this

view: the architecture is the most stationary aspect of the Internet. Everything else – the users, the devices, the applications – changes, and does so fairly rapidly, but the architecture stays the same. Therefore, solutions that are not inherently tied to the architecture (architectural "barnacles" (Anderson et al., 2005)) risk weighing it down as trends change. Is the temporary relief worth the added long-term complexity?

This is the tension that DDoS mitigation solutions from the literature have found themselves in for the past 20 years. To approach this challenge, we designed Gatekeeper to fit within the Internet ecosystem and architecture as much as possible. We do not propose new hardware, modifications to servers or clients, new wire protocols, or shared mechanisms between networks. Instead, we propose addressing the architectural aspects of DDoS using well-known techniques and readily-available infrastructure, and doing so while providing flexibility and performance, as well as aligning incentives, to achieve full deployability.

Central to this approach is the choice of vantage points, which help Gatekeeper cut to the heart of some of the architectural issues and drive deployment. Their distributed nature helps Gatekeeper combat the asymmetry of defending a single network against Internet-scale attacks, and provides topological insight that can help neutralize source address spoofing. Their proximity to source networks helps Gatekeeper enforce a connection-oriented network layer with network capabilities, while minimizing the amount of wasted resources. And the fact that they are well-provisioned is a natural fit for the DDoS mitigation use case, and lowers financial barriers to deployment.

We think that this approach is timely. Amid the rising magnitude, frequency, and sophistication of attacks, there exists a real possibility of large-scale link at-

tacks such as Crossfire becoming commonplace in the near future. Since Crossfire takes particular advantage of the architectural shortcomings, such as using legitimate-looking traffic and positioning the attack locus outside of the victim network's control, an approach is needed that neutralizes the architectural issues at play. By removing the need for a collaborative defense between networks and introducing path diversity to scramble an attacker's attempts at funneling traffic through congested links, Gatekeeper shows promise for combating these next-generation attacks as no other solution has; moreover, it does so *deployably*.

So, is Gatekeeper a barnacle? We suppose that depends on what the reader considers to be inside or outside of the architecture. Are IXPs and clouds extra-architectural? Does a connection-oriented network layer infringe on the openness property of the Internet? Do programs in the network core violate the end-to-end principle? We do not claim to have the exact answers to these questions, but we do feel comfortable foregoing architectural purity as necessary to mitigate the damage that DDoS attackers have been dealing to the Internet with impunity. We are also confident that leaving networks undefended and forced to pay for protective services was not an intended aspect of the architecture. So although Gatekeeper may or may not fit within the architecture as it was strictly *intended*, it certainly coexists with the architecture as it *is*. For us – and for the networks that have chosen to deploy Gatekeeper – that is enough.

BIBLIOGRAPHY

3GPP Org. (2019). 3GPP release 15. https://www.3gpp.org/release-15.

Akamai (2017). Q4 2017 State of the Internet / Security Report. https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q4-2017-state-of-the-internet-security-report.pdf.

Akamai (2019). State of the Internet / Security: 2019 – A Year in Review. https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/soti-security-a-year-in-review-report-2019.pdf.

Akamai (2020). Prolexic IP Protect. https://www.akamai.com/us/en/multimedia/documents/product-brief/prolexic-proxy-product-brief.pdf.

Amazon Web Services (2020). AWS IP address ranges. https://docs.aws.amazon.com/general/latest/gr/aws-ip-ranges.html.

Andersen, D. G. (2003). Mayday: Distributed filtering for Internet services. In *USENIX Symposium on Internet Technologies and Systems (USITS '03)*, vol. 4, (pp. 20–30).

Anderson, T., Peterson, L., Shenker, S., & Turner, J. (2005). Overcoming the Internet impasse through virtualization. *IEEE Computer*, *38*(4), 34–41.

Anderson, T., Roscoe, T., & Wetherall, D. (2004). Preventing Internet denial-of-service with capabilities. *ACM SIGCOMM Computer Communication Review (CCR)*, *34*(1), 39–44.

Anstee, D., Bowen, P., Chui, C., & Sockrider, G. (2017). 12^{th} Annual Worldwide Infrastructure Security Report. https://www.netscout.com/news/press-release/worldwide-infrastructure-security-report.

Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Invernizzi, L., Kallitsis, M., et al. (2017). Understanding the Mirai botnet. In *USENIX Security Symposium (SEC '17)*, (pp. 1093–1110).

Arazi, E. (2020). Why free DDoS protection can be the most expensive. https://blog.radware.com/security/ddos/2020/02/why-free-ddos-protection-can-be-the-most-expensive.

Arbor Networks (2014). 9^{th} Worldwide Infrastructure Security Report. https : //archive . nanog . org/sites/default/files/tuesday_general_sockrider_infrastructure_3.pdf.

Argyraki, K., & Cheriton, D. (2005a). Network capabilities: The good, the bad and the ugly. *ACM Hot Topics in Networks (HotNets-IV)*.

Argyraki, K. J., & Cheriton, D. R. (2005b). Active Internet traffic filtering: Real-time response to denial-of-service attacks. In *USENIX Annual Technical Conference (ATEC '05)*, (pp. 135–148).

AWS (2020a). AWS Shield Pricing. https://aws.amazon.com/shield/pricing.

AWS (2020b). AWS Shield Threat Landscape Report, Q1 2020. https://aws-shield-tlr.s3.amazonaws.com/2020-Q1_AWS_Shield_TLR.pdf.

AWS (2020c). Global Infrastructure. https://aws.amazon.com/about-aws/global-infrastructure.

AWS (2020d). Peering Policy. https://aws.amazon.com/peering/policy.

Barbette, T., Soldani, C., & Mathy, L. (2015). Fast userspace packet processing. In *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '15)*, (pp. 5–16).

Barr, J. (2016). Elastic Network Adapter – High Performance Network Interface for Amazon EC2. https://aws.amazon.com/blogs/aws/elastic-network-adapter-high-performance-network-interface-for-amazon-ec2.

Basescu, C., Reischuk, R. M., Szalachowski, P., Perrig, A., Zhang, Y., Hsiao, H., Kubota, A., & Urakawa, J. (2016). SIBRA: Scalable internet bandwidth reservation architecture. In *Network and Distributed System Security Symposium (NDSS '16)*.

BIRD (2020). The BIRD Internet Routing Daemon. https://bird.network.cz.

Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012). Fog computing and its role in the Internet of Things. In *ACM SIGCOMM Workshop on Mobile Cloud Computing (MCC '12)*, (pp. 13–16).

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., et al. (2014). P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review (CCR)*, *44*(3), 87–95.

Bray, H. (2016). Akamai breaks ties with security expert. https://www.bostonglobe.com/business/2016/09/23/cybercrooks-akamai/qOAhvHoohJcmkxIwg5ChKO/story.html.

Bright, P. (2013). Can a DDoS break the internet? Sure... just not all of it. https://arstechnica.com/information-technology/2013/04/can-a-ddos-break-the-internet-sure-just-not-all-of-it.

Buczak, A. L., & Guven, E. (2015). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, *18*(2), 1153–1176.

Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., & Shenker, S. (2007). Ethane: Taking control of the enterprise. *ACM SIGCOMM Computer Communication Review (CCR)*, *37*(4), 1–12.

Chatzis, N., Smaragdakis, G., Feldmann, A., & Willinger, W. (2013). There is more to IXPs than meets the eye. *ACM SIGCOMM Computer Communication Review (CCR)*, *43*(5), 19–28.

Cisco (2020). Cisco Annual Internet Report (20182023) White Paper. Tech. rep., Cisco.

Cisco NetFlow (2012). Introduction to Cisco IOS NetFlow - a technical overview. https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html.

Cloudflare (2020a). DDoS Protection with Cloudflare. https://www.cloudflare.com/resources/assets/slt3lc6tev37/2hIapovmEBdhDq0DLCuwDR/3691243ee090906900ba1a8dce7ddd45/Two_Pager_Rate_Limiting_Letter_EN-US.pdf.

Cloudflare (2020b). IP Ranges. https://www.cloudflare.com/ips.

Corero (2019). Infographic: Impact of DDoS on Enterprise Organizations. https://www.corero.com/blog/infographic-impact-of-ddos-on-enterprise-organizations/.

Cox, B. (2019). The speed of BGP network propagation. https://blog.apnic.net/2019/05/15/the-speed-of-bgp-network-propagation/.

Czyz, J., Allman, M., Zhang, J., Iekel-Johnson, S., Osterweil, E., & Bailey, M. (2014a). Measuring IPv6 adoption. *ACM SIGCOMM Computer Communications Review (CCR)*, *44*(4), 8798.

Czyz, J., Kallitsis, M., Gharaibeh, M., Papadopoulos, C., Bailey, M., & Karir, M. (2014b). Taming the 800 pound gorilla: The rise and decline of NTP DDoS attacks. In *ACM Internet Measurement Conference (IMC '14)*, (pp. 435–448).

Degraaf, R., Aycock, J., & Jacobson, M. (2005). Improved port knocking with strong authentication. In *IEEE Annual Computer Security Applications Conference (ACSAC '05)*.

Dhamdhere, A., & Dovrolis, C. (2010). The Internet is flat: modeling the transition from a transit hierarchy to a peering mesh. In *ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT '10)*, (pp. 1–12).

Doucette, C. (2020). https://github.com/cjdoucette/bird/tree/gatekeeper.

DPDK (2020). Intel Data Plane Development Kit (DPDK). http://dpdk.org.

Fabre, A. (2019). L4Drop: XDP DDoS Mitigations. https://blog.cloudflare.com/l4drop-xdp-ebpf-based-ddos-mitigations.

Firestone, D., Putnam, A., Mundkur, S., Chiou, D., Dabagh, A., Andrewartha, M., Angepat, H., Bhanu, V., Caulfield, A., Chung, E., et al. (2018). Azure accelerated networking: SmartNICs in the public cloud. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI '18)*, (pp. 51–66).

Floyd, S., Ramakrishnan, K., & Black, D. (2001). RFC 3168: The Addition of Explicit Congestion Notification (ECN) to IP. *Request for Comments, IETF*.

Fu, Q. (2020). *High Performance Software Packet Processing*. Ph.D. thesis, Boston University.

Gallenmüller, S., Emmerich, P., Wohlfart, F., Raumer, D., & Carle, G. (2015). Comparison of frameworks for high-performance packet IO. In *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '15)*, (pp. 29–38).

Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., & Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *Elsevier Computers & Security*, *28*(1-2), 18–28.

Geng, X., & Whinston, A. B. (2000). Defeating distributed denial of service attacks. *IEEE IT Professional*, *2*(4), 36–42.

Gershuni, E., Amit, N., Gurfinkel, A., Narodytska, N., Navas, J. A., Rinetzky, N., Ryzhyk, L., & Sagiv, M. (2019). Simple and precise static analysis of untrusted Linux kernel extensions. In *CM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '19)*, (pp. 1069–1084).

Gilad, Y., Herzberg, A., Sudkovitch, M., & Goberman, M. (2016). CDN-on-Demand: An affordable DDoS defense via untrusted clouds. In *Network and Distributed System Security Symposium (NDSS '16)*.

Gill, P., Arlitt, M., Li, Z., & Mahanti, A. (2008). The flattening internet topology: Natural evolution, unsightly barnacles or contrived collapse? In *International Conference on Passive and Active Network Measurement (PAM '08)*, (pp. 1–10).

Gillani, F., Al-Shaer, E., Lo, S., Duan, Q., Ammar, M., & Zegura, E. (2015). Agile virtualized infrastructure to proactively defend against cyber attacks. In *IEEE Conference on Computer Communications (INFOCOM '15)*, (pp. 729–737).

Giotsas, V., Dhamdhere, A., & Claffy, K. C. (2016). Periscope: Unifying looking glass querying. In *International Conference on Passive and Active Network Measurement (PAM '16)*, (pp. 177–189).

Giotsas, V., Smaragdakis, G., Dietzel, C., Richter, P., Feldmann, A., & Berger, A. (2017). Inferring BGP blackholing activity in the Internet. In *ACM Internet Measurement Conference (IMC '17)*, (pp. 1–14).

Google (2020a). Statistics: IPv6 adoption. https://www.google.com/intl/en/ipv6/statistics.html.

Google (2020b). Where can I find Compute Engine IP ranges? https://cloud.google.com/compute/docs/faq#find_ip_range.

Guirguis, M., Bestavros, A., & Matta, I. (2004). Exploiting the transients of adaptation for RoQ attacks on Internet resources. In *IEEE International Conference on Network Protocols (ICNP '04)*, (pp. 184–195).

Guo, H., & Heidemann, J. (2020). Detecting IoT devices in the internet. *IEEE/ACM Transactions on Networking (TON '20)*, (pp. 2323–2336).

Gupta, R., Tanwar, S., Tyagi, S., & Kumar, N. (2019). Tactile-Internet-based telesurgery system for healthcare 4.0: An architecture, research challenges, and future directions. *IEEE Network*, *33*(6), 22–29.

Han, B., Gopalakrishnan, V., Ji, L., & Lee, S. (2015). Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, *53*(2), 90–97.

Han, D., Anand, A., Dogar, F., Li, B., Lim, H., Machado, M., Mukundan, A., Wu, W., Akella, A., Andersen, D. G., et al. (2012). XIA: Efficient support for evolvable internetworking. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI '12)*, (pp. 309–322).

Han, S., Jang, K., Park, K., & Moon, S. (2010). PacketShader: a GPU-accelerated software router. *ACM SIGCOMM Computer Communication Review (CCR)*, *40*(4), 195–206.

Haq, O., Doucette, C., Byers, J. W., & Dogar, F. R. (2020). Judicious QoS using cloud overlays. In *ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT '20)*, (pp. 371–385).

Haq, O., Raja, M., & Dogar, F. R. (2017). Measuring and improving the reliability of wide-area cloud paths. In *International Conference on World Wide Web (WWW '17)*, (pp. 253–262).

Hasan, S., Gorinsky, S., Dovrolis, C., & Sitaraman, R. K. (2014). Trade-offs in optimizing the cache deployments of CDNs. In *IEEE Conference on Computer Communications (INFOCOM '14)*, (pp. 460–468).

Hsiao, H.-C., Kim, T. H.-J., Yoo, S., Zhang, X., Lee, S. B., Gligor, V., & Perrig, A. (2013). STRIDE: sanctuary trail – refuge from Internet DDoS entrapment. In *ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIA CCS '13)*, (pp. 415–426).

Ierusalimschy, R., De Figueiredo, L. H., & Celes, W. (2006). Lua 5.1 reference manual. https://www.lua.org/manual/5.1.

Imperva (2020). Imperva DDoS Protection. https://www.imperva.com/products/ddos-protection-solutions/.

IX.br (2020). IX.br aggregate statistics. https://ix.br/agregado.

Iyengar, S. (2018). Moving fast at scale: Experience deploying IETF QUIC at Facebook. In *ACM CoNEXT Workshop on the Evolution, Performance, and Interoperability of QUIC (EPIQ '18)*, (p. Keynote).

Kang, M. S., Gligor, V. D., Sekar, V., et al. (2016). SPIFFY: Inducing cost-detectability tradeoffs for persistent link-flooding attacks. In *Network and Distributed System Security Symposium (NDSS '16)*.

Kang, M. S., Lee, S. B., & Gligor, V. D. (2013). The crossfire attack. In *IEEE Symposium on Security and Privacy (S&P '13)*, (pp. 127–141).

Konte, M., Perdisci, R., & Feamster, N. (2015). ASwatch: An AS reputation system to expose bulletproof hosting ASes. In *ACM SIGCOMM*, (pp. 625–638).

Krebs, B. (2016). The democratization of censorship. https://krebsonsecurity.com/2016/09/the-democratization-of-censorship.

Krishnan, P., Raz, D., & Shavitt, Y. (2000). The cache location problem. *IEEE/ACM Transactions on Networking (TON '00)*.

Lakhina, A., Crovella, M., & Diot, C. (2005). Mining anomalies using traffic feature distributions. *ACM SIGCOMM Computer Communication Review (CCR)*, *35*(4), 217–228.

Langley, A., Riddoch, A., Wilk, A., Vicente, A., Krasic, C., Zhang, D., Yang, F., Kouranov, F., Swett, I., Iyengar, J., et al. (2017). The QUIC transport protocol: Design and Internet-scale deployment. In *ACM SIGCOMM*, (pp. 183–196).

Lee, S. B., Kang, M. S., & Gligor, V. D. (2013). CoDef: Collaborative defense against large-scale link-flooding attacks. In *ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT '13)*, (pp. 417–428).

Liu, X., Yang, X., & Lu, Y. (2008). To filter or to authorize: Network-layer DoS defense against multimillion-node botnets. In *ACM SIGCOMM*, (pp. 195–206).

Liu, X., Yang, X., & Xia, Y. (2010). NetFence: preventing Internet denial of service from inside out. In *ACM SIGCOMM*, (pp. 255–266).

Liu, Z., Jin, H., Hu, Y.-C., & Bailey, M. (2016). MiddlePolice: Toward enforcing destination-defined policies in the middle of the Internet. In *ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*, (pp. 1268–1279).

Lyon, G. F. (2009). *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure.

Machado, M., Doucette, C., & Byers, J. W. (2015). Linux XIA: an interoperable meta network architecture to crowdsource the future Internet. In *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '15)*, (pp. 147–158). IEEE.

Machado, M., Doucette, C., & Fu, Q. (2020). Gatekeeper. https://github.com/AltraMayor/gatekeeper.

Mahajan, R., Bellovin, S. M., Floyd, S., Ioannidis, J., Paxson, V., & Shenker, S. (2002). Controlling high bandwidth aggregates in the network. *ACM SIGCOMM Computer Communication Review (CCR)*, *32*(3), 62–73.

Mahindra, R., Chandar, V., & Guo, E. (2019). Employing QUIC protocol to optimize Ubers app performance. https://eng.uber.com/employing-quic-protocol.

Majkowski, M. (2017). Meet Gatebot - a bot that allows us to sleep. https://blog.cloudflare.com/meet-gatebot-a-bot-that-allows-us-to-sleep.

Majkowski, M. (2018). How to drop 10 million packets per second. https://blog.cloudflare.com/how-to-drop-10-million-packets/.

Makrushin, D. (2017). The cost of launching a DDoS attack. https://securelist.com/analysis/publications/77784/the-cost-of-launching-a-ddos-attack.

Marvin, R. (2019). Chinese DDoS attack hits Telegram during Hong Kong protests. https://www.pcmag.com/news/chinese-ddos-attack-hits-telegram-during-hong-kong-protests.

MaxMind, I. (2020). GeoIP Products. https://dev.maxmind.com/geoip.

McCanne, S., & Jacobson, V. (1993). The BSD Packet Filter: A New Architecture for User-level Packet Capture. In *USENIX Winter*, vol. 46.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., & Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review (CCR)*, *38*(2), 69–74.

Meier, R., Tsankov, P., Lenders, V., Vanbever, L., & Vechev, M. (2018). NetHide: Secure and practical network topology obfuscation. In *USENIX Security Symposium (SEC '18)*, (pp. 693–709).

Menscher, D. (2020). Exponential growth in DDoS attack volumes. https : //cloud . google . com/blog/products/identity - security/identifying - and-protecting-against-the-largest-ddos-attacks.

Moore, D., Voelker, G. M., & Savage, S. (2001). Inferring internet denial-of-service activity. *USENIX Security Symposium (SEC '01)*.

Nathan, A. (2020). DRIB. https://github.com/andrenth/dribf.

NERD (2020). Network entity reputation database. https://nerd.cesnet.cz/.

NETSCOUT (2019). Cloud in the crosshairs. NETSCOUT's 14th annual worldwide infrastructure security report. https://www.netscout.com/report.

Neustar (2020). DDoS attacks increase 180% in 2019 compared to 2018. https://www.home.neustar/about-us/news-room/press-releases/2020/ddos-attacks-increase-180-in-2019-compared-to-2018.

NexusGuard (2020). NexusGuard DDoS threat report, 2020 Q1. https://blog.nexusguard.com/threat-report/ddos-threat-report-2020-q1.

Nygren, E., Sitaraman, R. K., & Sun, J. (2010). The Akamai network: a platform for high-performance Internet applications. *ACM SIGOPS Operating Systems Review (OSR)*, *44*(3), 2–19.

Packet Clearing House (2020). Internet exchange point growth by country. https://www.pch.net/ixp/summary_growth_by_country.

Packetbeat (2020). Lightweight shipper for network data. https://www.elastic.co/beats/packetbeat.

Parno, B., et al. (2007). Portcullis: protecting connection setup from denial-of-capability attacks. In *ACM SIGCOMM*, (pp. 289–300).

Patternson, D. (2015). Exclusive: Inside the ProtonMail siege: how two small companies fought off one of Europe's largest DDoS attacks. https://www.techrepublic.com/article/exclusive-inside-the-protonmail-siege-how-two-small-companies-fought-off-one-of-europes-largest-ddos/.

Paxson, V. (2001). An analysis of using reflectors for distributed denial-of-service attacks. *ACM SIGCOMM Computer Communication Review (CCR)*, *31*(3), 38–47.

PeeringDB (2020). The interconnection database. https://www.peeringdb.com/.

PF_RING (2020). PF_RING: high-speed packet capture, filtering and analysis. https://www.ntop.org/products/packet-capture/pf_ring/.

Prince, M. (2013). The DDoS that knocked Spamhaus offline (and how we mitigated it). https://blog.cloudflare.com/the-ddos-that-knocked-spamhaus-offline-and-ho/.

Redis (2020). https://redis.io/.

Rizzo, L. (2012). Netmap: a novel framework for fast packet I/O. In *USENIX Security Symposium (SEC '12)*, (pp. 101–112).

Rossow, C. (2014). Amplification hell: Revisiting network protocols for DDoS abuse. In *Network and Distributed System Security Symposium (NDSS '14)*.

Satyanarayanan, M. (2017). The emergence of edge computing. *Computer*, *50*(1), 30–39.

SeattleIX (2020). Traffic graphs. https://www.seattleix.net/statistics.

Shi, L., Sisodia, D., Zhang, M., Li, J., Dainotti, A., & Reiher, P. (2019). The catch-22 attack. In *IEEE Annual Computer Security Applications Conference (ACSAC '19)*.

Shirokov, N., & Dasineni, R. (2018). Open-sourcing Katran, a scalable network load balancer. https://engineering.fb.com/open-source/open-sourcing-katran-a-scalable-network-load-balancer/.

Smith, J. M., & Schuchard, M. (2018). Routing around congestion: Defeating DDoS attacks and adverse network conditions via reactive BGP routing. In *IEEE Symposium on Security and Privacy (S&P '18)*, (pp. 599–617).

Studer, A., & Perrig, A. (2009). The coremelt attack. In *European Symposium on Research in Computer Security (ESORICS '09)*, (pp. 37–52).

Team Cymru (2020). The Bogon Reference. https://team-cymru.com/community-services/bogon-reference/.

The Spamhaus Project SLU (2020). The Spamhaus Don't Route Or Peer Lists. https://www.spamhaus.org/drop/.

Tran, M., Kang, M. S., Hsiao, H.-C., Chiang, W.-H., Tung, S.-P., & Wang, Y.-S. (2019). On the feasibility of rerouting-based DDos defenses. In *IEEE Symposium on Security and Privacy (S&P '19)*, (pp. 1169–1184).

Tuerk, M. (2020). Africa Is The Next Frontier For The Internet. https://www.forbes.com/sites/miriamtuerk/2020/06/09/africa-is-the-next-frontier-for-the-internet/?sh=b6d309b49001.

Venkatesan, S., Albanese, M., Amin, K., Jajodia, S., & Wright, M. (2016). A moving target defense approach to mitigate DDoS attacks against proxy-based architectures. In *IEEE Conference on Communications and Network Security (CNS '16)*, (pp. 198–206).

Wang, H., Jia, Q., Fleck, D., Powell, W., Li, F., & Stavrou, A. (2014). A moving target DDoS defense mechanism. *Computer Communications*, *46*, 10–21.

Wetherall, D. J., Guttag, J. V., & Tennenhouse, D. L. (1998). ANTS: A toolkit for building and dynamically deploying network protocols. In *1998 IEEE Open Architectures and Network Programming*, (pp. 117–129).

World IPv6 Launch (2020). Network operator measurements. https://www.worldipv6launch.org/measurements/.

Wragg, D. (2020). Unimog - Cloudflares edge load balancer. https://blog.cloudflare.com/unimog-cloudflares-edge-load-balancer/.

Xu, D., Zhou, A., Zhang, X., Wang, G., Liu, X., An, C., Shi, Y., Liu, L. L., & Ma, H. (2020). Understanding operational 5G: A first measurement study on its coverage, performance and energy consumption. In *ACM SIGCOMM*, (pp. 479–494).

Xue, L., Luo, X., Chan, E. W., & Zhan, X. (2014). Towards detecting target link flooding attack. In *USENIX Large Installation System Administration Conference (LISA '14)*, (pp. 90–105).

Yaar, A., Perrig, A., & Song, D. (2004). SIFF: A stateless internet flow filter to mitigate DDoS flooding attacks. In *IEEE Symposium on Security and Privacy (S&P '04)*, (pp. 130–143).

Yang, X., Wetherall, D., & Anderson, T. (2005). A DoS-limiting network architecture. In *ACM SIGCOMM*, (pp. 241–252).

Yang, Z., Cui, Y., Li, B., Liu, Y., & Xu, Y. (2019). Software-defined wide area network (SD-WAN): Architecture, advances and opportunities. In *IEEE International Conference on Computer Communication and Networks (ICCCN '19)*, (pp. 1–9).

Yap, K.-K., Motiwala, M., Rahe, J., Padgett, S., Holliman, M., Baldus, G., Hines, M., Kim, T., Narayanan, A., Jain, A., et al. (2017). Taking the edge off with espresso: Scale, reliability and programmability for global internet peering. In *ACM SIGCOMM*, (pp. 432–445).

Yeganeh, B., Durairajan, R., Rejaie, R., & Willinger, W. (2019). How cloud traffic goes hiding: A study of Amazon's peering fabric. In *ACM Internet Measurement Conference (IMC '19)*, (pp. 202–216).

Zhang, X., Hsiao, H.-C., Hasker, G., Chan, H., Perrig, A., & Andersen, D. G. (2011). Scion: Scalability, control, and isolation on next-generation networks. In *IEEE Symposium on Security and Privacy (S&P '11)*, (pp. 212–227).

Zheng, J., Li, Q., Gu, G., Cao, J., Yau, D. K., & Wu, J. (2018). Realtime DDoS defense using COTS SDN switches via adaptive correlation analysis. *IEEE Transactions on Information Forensics and Security (TIFS '18)*, *13*(7), 1838–1853.

CURRICULUM VITAE

Cody Doucette

January 6, 2021

117 Mount Blue St.
Norwell, MA 02061
cody.j.doucette@gmail.com

111 Cummington Mall
Boston University
Boston, MA 02215
doucette@bu.edu

Academic Training:

01/2021	PhD Boston University, Boston, MA; Computer Science
05/2014	MS Boston University, Boston, MA; Computer Science
05/2014	BA Boston University, Boston, MA; Computer Science

Doctoral Research:

Title:	An Architectural Approach for Mitigating Next-Generation Denial of Service Attacks
Thesis advisor:	John W. Byers, PhD
Defense date:	December 3, 2020
Summary:	My dissertation presents the design, implementation, and evaluation of *Gatekeeper*, an architecture-aware, deployable DDoS mitigation service.

Original, Peer Reviewed Publications (newest first):

1. Osama Haq, Cody Doucette, John W. Byers, Fahad R. Dogar. Judicious QoS using Cloud Overlays. *The 16th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. 2020.

2. Michel Machado, Cody Doucette, and John W. Byers. Linux XIA: an interoperable meta network architecture to crowdsource the future Internet. *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. 2015.

3. David Naylor et al. XIA: architecting a more trustworthy and evolvable internet. *ACM SIGCOMM Computer Communication Review (CCR)*. 2014.